Abstract

Shortcuts are decision rules that exploit spurious correlations between the input attributes and labels that hold for the majority of the training examples. Consequently, they lead to worse performance on minority data groups where the spurious correlations do not hold. Models fine-tuned with Empirical Risk Minimization (ERM) have been observed to struggle with predictions on out-of-distribution (OOD) test sets, where the proportion of data groups differs from the one seen by the model during training, due to the model's reliance on shortcuts. In response, previous research has proposed various modifications to the loss function to re-weight the contribution of training examples; however, the effect of these modifications on the models' internal representations and decision-making processes is not well understood. In this thesis, we develop **Competing Rules Hypothesis (CRH)**, which describes the model's decision-making process as a competition between an intended rule and a shortcut rule, as a framework to understand how models implement simple shortcuts. Building on CRH, we propose **Representation Shift (REPRSHIFT)**, which surgically modifies a single layer inside the network to systematically shift the representations of examples with shortcuts, as an interpretability-based approach to shortcut mitigation.

Our experiments are divided into two parts. In the first part, we inspect the internal representations learned by models fine-tuned with ERM and four existing loss function-based shortcut mitigation methods. Using natural language inference (MultiNLI) and toxicity detection (CivilComments) datasets, we (1) compare the representations learned by different loss functions using Centered Kernel Alignment (CKA), (2) probe the representations for information about the shortcut attributes, and (3) investigate how the classifier layers use the information from the earlier layer representations to make predictions. In the second part, we conduct causal intervention experiments to understand how an ERM-trained model implements a shortcut rule and find suggestive evidence for CRH. Finally, we demonstrate that REPRSHIFT can be used to substantially improve worst-group performance on MultiNLI.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

1.1 Motivation

In recent years, pre-trained Transformers, such as BERT (Devlin et al., 2019) and GPT (Radford and Narasimhan, 2018; Radford et al., 2019), have lead to remarkable advances in natural language processing (Kalyan et al., 2021; Min et al., 2021; Periti et al., 2024). Meanwhile, fine-tuning pre-trained transformers emerged as the state-of-the-art method for text classification benchmarks (Cortiz, 2021; Devlin et al., 2019; Li et al., 2021; Liu et al., 2019; Sanh et al., 2020; Yang et al., 2020; Yu et al., 2023). These models are increasingly being used in high-stakes applications. Natural Language Inference (NLI), which determines the logical relationship between two sentences, and toxicity detection, which identifies whether a text contains harmful or offensive language, are text classification tasks with important applications to content moderation on social platforms. Accurate and reliable models can be used to fact-check statements (Thorne et al., 2018) and detect harmful comments (Borkan et al., 2019) to improve the quality of online discourse. While the recent advances have led to significant improvements in accuracy, two notable challenges remain on the the reliability front:

1. The performance of models drops significantly when evaluated on specific subsets of examples (data groups), which also results in poor overall performance when the models are evaluated on datasets with different data group proportions (out-of-distribution datasets).

2. The internal decision-making processes of models are not well understood, which undermines their safe deployment in sensitive domains.

This motivates our investigation at the intersection of these challenges. We explore the internal decision-making processes of models to understand the cause of the significant

performance discrepancies across different subsets of examples. Based on our understanding, we demonstrate that we can surgically edit a single layer in the model to modify the representations and improve the model's worst-group performance.

1.2 Problem

Shortcuts are decision rules that exploit spurious correlations between the input attributes and labels that hold for the majority of the training examples (Geirhos et al., 2020). They lead to worse performance on minority data groups where the spurious correlations do not hold, which also results in poor overall performance when the models are evaluated on datasets with different data group proportions. Following the previous literature (Geirhos et al., 2020; Yang et al., 2023), we define a data group as a subset of the dataset that contains examples with a specific pattern that is not causally related to the label (an attribute) and the same label.

Attribute	**Label**	**Premise**	**Hypothesis**
Negation Word $a = 1$	Contradiction $y = 0$	the mausoleum was designed in florence, as a gift from the grand duke of tuscany.	the grand duke of tuscany never gave anyone gifts for any reason.
	Entailment $y = 1$	only the right arm of one of the two transepts and the octagonal belltower [...] remain.	the left arms of the two transepts no longer remain.
	Neutral $y = 2$	uh yeah except i don't use them	i have never found the need to use them.

Table 1.1 Shortcut Attribute Examples from MultiNLI (Williams et al., 2018)

For example, the objective of an NLI task is to predict whether a premise contradicts, entails, or is neutral to a hypothesis. Table 1.1 demonstrates three examples from the MultiNLI (Williams et al., 2018) dataset that have a specific pattern (an attribute): the hypothesis contains the negation words "no", "never", "nobody" or "nothing" (Idrissi et al., 2022). A premise-hypothesis pair where the hypothesis contains a negation word can have any of the three labels (Contradiction, Entailment, or Neutral) based on the logical relationship between the two sentences. However, as demonstrated in Table 1.2, in the MultiNLI dataset, most of the examples where the hypotheses contain negation words have contradiction labels. Based on our definition of data groups as attribute-label pairs, we

	Contradiction	**Entailment**	**Neutral**
Overall - $P(y)$	33.3%	33.4%	33.3%
No Negation - $P(y\|a=0)$	30.0%	35.2%	34.8%
Negation - $P(y\|a=1)$	76.1%	10.4%	13.6%

Table 1.2 MultiNLI Spurious Correlations. (Idrissi et al., 2022)

observe that the data group *Negation Word ($a=1$) & Contradiction ($y=0$)* is overrepresented in the MultiNLI dataset compared to *Negation Word ($a=1$) & Entailment ($y=1$)* and *Negation Word ($a=1$) & Neutral ($y=2$)* groups (Idrissi et al., 2022). Under this setup, the MultiNLI dataset is divided into 6 data groups, which, in addition to the 3 groups demonstrated in Table 1.1, also include 3 groups where the hypotheses do not contain any of the four negation words.

	No Negation Word			Negation Word		
	Contradiction	Entailment	Neutral	Contradiction	Entailment	Neutral
Predicted Contradiction	**82.0**	5.8	11.3	**94.7**	11.0	28.3
Predicted Entailment	6.6	**83.7**	10.9	0.9	**76.9**	8.9
Predicted Neutral	11.5	10.5	**77.8**	4.5	12.1	**62.8**

Table 1.3 MultiNLI Per Attribute Confusion Matrix (Test). 6 columns indicate the 6 data groups, and the rows are predicted labels. The values inside the cells are the percentage of examples that are predicted as contradiction, entailment, or neutral for each group. Hence, the column sums add up to 100%. The reported results are the averages on the test set from 3 fine-tuning runs. ERM is used as the fine-tuning loss function.

What happens when we fine-tune a model on MultiNLI using Empirical Risk Minimization (ERM, a loss function that equally weights each example in the training set)? The result is demonstrated in Table 1.3, where the accuracy for each group is shown in highlighted cells. While the model performs reasonably well on examples where the hypothesis does not contain a negation word, the presence of a negation word significantly impacts the model's performance: It improves the accuracy for examples labeled as contradiction 12.7% at the expense of the accuracy of examples labeled as neutral, for which the accuracy drops 15%.

1.2.1 Out of Distribution Generalization

Our model from Table 1.3 performs well in evaluation on datasets where most of the examples that contain negation words are labeled as a contradiction, which is a pattern that holds true in the MultiNLI dataset overall. When we randomly split the dataset into training and test

sets, the overall trends in the dataset are reflected in the splits as well. As a result, our test set comes from the same distribution as the training set, i.e., it is an independent and identically distributed (IID) test set. In this test set, where our model achieves an overall accuracy of 81.2%, the majority of the examples that contain negation words also have contradiction labels. The overall accuracy is the weighted sum of the group accuracies, where the weight of each data group is the percentage of examples in the dataset that belong to the group. Hence, under distribution shifts, when the proportion of data groups changes, the overall performance of the model also shifts accordingly. In a test set where there are many examples that belong to the group *Negation Word* ($a = 1$) *&* *Contradiction* ($y = 0$), the model's performance would approach 94.7%. Similarly, the overall performance of the model would get closer to 62.8% as the proportion of examples that belong to group *Negation Word* ($a = 1$) *&* *Neutral* ($y = 2$). A test set is called out-of-distribution (OOD) when its proportion of data groups is different from the proportion of data groups in the training set. The accuracy of the model on the worst-performing data groups gives a lower bound estimate for model performance in OOD settings, where the proportion of defined data groups can change arbitrarily.

1.3 Loss-Function-Based Approaches

There is extensive literature that studies training models that perform well on OOD test sets, where the examples are sampled from a different distribution during evaluation than the one on which the model is trained (Cai et al., 2021; Creager et al., 2021; DeGrave et al., 2021; Du et al., 2023; Geirhos et al., 2020; Gowda et al., 2021; Han et al., 2023; Idrissi et al., 2022; Izmailov et al., 2022; Joshi et al., 2022; Koh et al., 2021; Li et al., 2022; Liu et al., 2021; Nam et al., 2022; Sagawa et al., 2020; Yang et al., 2023; Yao et al., 2022). In the classical learning setting, called Empirical Risk Minimization (ERM) (Vapnik, 1999), the model parameters are updated based on the gradients of a loss function that equally weights the examples seen during training. The existing methods that aim to improve the performance of the models on OOD test sets propose different modifications to this loss function. For example, Group Distributionally Robust Optimization (GroupDRO) (Sagawa et al., 2020) proposes to minimize the expected loss for the data group that has the highest loss, whereas other methods do not assume knowledge of the data groups during training and assign weights to examples based on the difficulty of the example from the perspective of the model. In our first set of experiments, we examine the internal representations of models fine-tuned with various modifications to the loss function.

1.4 Interpretability Based Approach (Ours)

1.4.1 Competing Rules Hypothesis (CRH)

We propose the Competing Rules Hypothesis (CRH) as a framework to interpret the internal processes of a model that exhibits a simple shortcut learning behavior. CRH describes the model's decision-making process as a competition between an intended rule and a shortcut rule. The shortcut rule exploits spurious correlations to boost the model's performance for majority groups. The intended rule implements an algorithm that generalizes across different data groups.

Algorithm 1 Shortcut Rule (example)

```
if Hypothesis contains a negation word then
    Increase contradiction logit, decrease others
end if
```

Algorithm 2 Intended Rule (example)

```
if Premise contradicts hypothesis then
    Increase contradiction logit, decrease others
else if Premise entails hypothesis then
    Increase entailment logit, decrease others
else
    Increase neutral logit, decrease others
end if
```

Under this framework, Algorithms 1 and 2 describe the shortcut rule and intended rule implemented by a model trained on MultiNLI. Our observations suggest that the shortcut rule and the intended rule compete to determine the example's predicted label. This competition takes place in the logit space, the 3-dimensional output of the model, where each dimension determines the model's preference for one of the labels. The shortcut rule increases the logit corresponding to the contradiction label whenever there is a negation word in the hypothesis. The intended rule affects the logits based on an algorithm that accounts for the logical relationship between the two sentences. The model predicts the label that has the highest logit. We present three lines of suggestive evidence for CRH in Sections 3.2 and 4.4.1.

1.4.2 Representation Shift (REPRSHIFT)

Building on CRH, we propose an interpretability-based approach to shortcut mitigation. We reason that if the shortcut rule and the intended rule are competing in the logit space for

influence on the model's predictions, then *suppressing the shortcut rule* would allow the intended rule to dominate the predictions and improve the generalization of the model across data groups. To suppress the shortcut rule, we add an "Inverse Shortcut" rule into the model. Our reasoning can be described with simple **model algebra**:

We want to have a model with only INTENDED RULE
We have a model with INTENDED RULE + SHORTCUT RULE
We compute INVERSE SHORTCUT RULE = − SHORTCUT RULE

We add inverse shortcut rule to our model and get a model with
INTENDED RULE + SHORTCUT RULE + INVERSE SHORTCUT RULE

This is equal to
INTENDED RULE + SHORTCUT RULE + (− SHORTCUT RULE)

Shortcut rules cancel out and we get a model with
INTENDED RULE

For our NLI dataset, the inverse shortcut rule is

Algorithm 3 Inverse Shortcut Rule (informal)

1: **if** Hypothesis contains a negation word **then**
2: Decrease contradiction logit, increase others
3: **end if**

Our method, REPRSHIFT, surgically modifies a single layer inside an ERM-trained network to shift the representations of examples with shortcuts. We implement REPRSHIFT in 4 steps:

1. We compute an internal representation, k_{shortcut}, for the attribute that triggers the shortcut rule. We use this representation to implement the **if** statement in the inverse shortcut rule (Algorithm 3, Line 1).

2. We compute a logit shift vector, $v_{\text{logitshift}}$, that quantifies the impact of the shortcut rule on the logits.

3. We use this logit shift vector to compute a representation shift vector, $v_{\text{reprshift}}$, that implements the **then** statement in the inverse shortcut rule (Algorithm 3, Line 2).

4. Finally, we modify a single MLP weight matrix W inside our model to implement the inverse shortcut rule:

$$\textbf{if } k_{\text{shortcut}}, \textbf{ then } v_{\text{reprshift}} \quad \text{or, equivalently,} \quad k_{\text{shortcut}} \cdot W = v_{\text{reprshift}}$$

In Step 4, to modify W, we use the model editing approach introduced by Bau et al. (2020) with the closed form solution derived by Meng et al. (2023a). In Section 4.4.2, we show that REPRSHIFT can be used to substantially improve the worst group accuracy of an ERM-trained model on MultiNLI.

1.5 Contents

Background

Chapter 2 provides the background on distribution shifts and loss-function-based approaches. Firstly, in Section 2.1, we formally define different types of distribution shifts. Then, in Section 2.2, we introduce the loss-function-based approaches that aim to address distribution shifts.

Methodology

We begin Chapter 3 by explaining the internal processes of our model that compute the internal representations (Section 3.1). In Section 3.2, we introduce CRH and discuss two lines of suggestive evidence that motivate our formulation. In Section 3.3, we present REPRSHIFT.

Experiments

Chapter 4 presents our experiments in two parts. In the first part, we inspect the internal representations of models fine-tuned with ERM and four different loss-function-based shortcut mitigation methods. We use natural language inference (MultiNLI) and toxicity detection (CivilComments) datasets to conduct 3 experiments.

1. In Section 4.3.1, we compare the learned representations using Centered Kernel Alignment (CKA) (Kornblith et al., 2019) and find that the representations from different fine-tuning methods differ from the representations of the pre-trained model in the later layers of the network, but are similar to each other across all layers in MultiNLI.

2. In Section 4.3.2, we probe the representations for information about the shortcut attributes, and observe that the shortcut attribute probes achieve similar accuracies for ERM and loss-function-based shortcut mitigation methods.

3. In Section 4.3.3, we investigate how the classifier layers use the information from the earlier layers to make predictions and note that the model starts treating examples that contain shortcut attributes differently in the middle layers.

In the second part, we focus on models fine-tuned with ERM on our natural language inference dataset.

1. In Section 4.4.1, we conduct causal intervention experiments to understand how our model implements a shortcut rule and find a third line of suggestive evidence for CRH.

2. In Section 4.4.2, we evaluate REPRSHIFT against the loss-function-based approaches.

Conclusion

Chapter 5 concludes our work.

Chapter 2

Background

2.1 Distribution Shifts

Distribution shifts have been studied under domain adaptation (Quionero-Candela et al., 2009; Saenko et al., 2010), subpopulation shifts (Cai et al., 2021; Yang et al., 2023), and domain generalization (Koh et al., 2021). The objective of these studies is to train models that perform well when the examples are sampled from a different distribution during evaluation than the one on which the model is trained (Creager et al., 2021; DeGrave et al., 2021; Geirhos et al., 2020; Gowda et al., 2021; Han et al., 2023; Idrissi et al., 2022; Izmailov et al., 2022; Joshi et al., 2022; Koh et al., 2021; Li et al., 2022; Liu et al., 2021; Nam et al., 2022; Sagawa et al., 2020; Yang et al., 2023; Yao et al., 2022). Common types of distribution shifts include (a) spurious correlations, where examples that have a particular attribute a_i are more likely to have certain label y_i in the training set, (b) attribute imbalance, where the examples with a particular attribute a_i are underrepresented in the training set, (c) label imbalance, where the examples with a particular label y_i are underrepresented in the training set, and (d) attribute generalization, where examples with a certain attribute a_i are completely missing in the training set (Yang et al., 2023). We formally define these distribution shifts in the following sections, providing examples from the MultiNLI dataset where we have 2 attributes (a_0 and a_1) and 3 labels (y_0,y_1, and y_2), which make up 6 data groups.

2.1.1 Spurious Correlations

Under spurious correlations, the distribution of labels conditional on an attribute is skewed. For example, in MultiNLI, the examples that contain a negation word in the second sentence

(a_1) are more likely to have label contradiction (y_0).

$$P_{\text{IID}}(y_0|a_1) > P_{\text{IID}}(y_0|a_0), \tag{2.1}$$

where P_{IID} denotes the IID test distribution that matches the training distribution. Consequently, a model, f, trained on examples from this distribution can assign a higher probability to label y_0 for examples with attribute a_1.

$$P_f(y_0|a_1,x) > P_f(y_0|a_0,x), \tag{2.2}$$

where P_f denotes the probability distribution learned by the model and x denotes the input features. In the OOD test set, we may have

$$P_{\text{OOD}}(y_0|a_1) = P_{\text{OOD}}(y_0|a_0) \implies \text{Accuracy}_{\text{OOD}}(f) < \text{Accuracy}_{\text{IID}}(f), \tag{2.3}$$

where P_{OOD} denotes OOD test distribution. Under this setting, a model that relies on a_1 to inform the predictions of y_0 would experience drops in performance in OOD test sets. If certain input features, x_R, mediate the relationship between a_1 and y_0, conditioning the probabilities on x_R can result in a conditional independence between a_1 and y_0 given x_R:

$$P_{\text{IID}}(y_0|a_1,x_R) = P_{\text{IID}}(y_0|a_0,x_R) \tag{2.4}$$

2.1.2 Attribute Imbalance

Under attribute imbalance, certain attributes are less frequent in the training distribution compared to the test distribution. For instance, in MultiNLI, the examples that contain a negation word in the hypothesis are less frequent compared to the examples that do not contain a negation word in the hypothesis.

$$P_{\text{IID}}(a_1) < P_{\text{IID}}(a_0) \tag{2.5}$$

The models trained on datasets with attribute imbalance can have lower confidence for examples with underrepresented attributes.

$$P_f(y|a_1,x) > P_f(y|a_0,x) \tag{2.6}$$

Then, in OOD test sets, if the examples that the model is less confident about are more frequent, this can lead to overall performance drops.

$$P_{\text{OOD}}(a_1) = P_{\text{OOD}}(a_0) \implies \text{Accuracy}_{\text{OOD}}(f) < \text{Accuracy}_{\text{IID}}(f) \tag{2.7}$$

2.1.3 Label Imbalance

Similarly, under label imbalance, certain labels are less frequent in the training distribution compared to the test distribution. In MultiNLI, we have a balanced label distribution; however, in the CivilComments dataset, the comments that are labeled as non-toxic are much more frequent compared to those that are labeled as toxic.

$$P_{\text{IID}}(y_0) > P_{\text{IID}}(y_1) \tag{2.8}$$

The models trained on datasets with label imbalance can have lower preference for examples with underrepresented labels independent of the input features.

$$P_f(y_0) > P_f(y_1) \tag{2.9}$$

In OOD settings where the test set has a balanced label distribution, this can lead to overall performance drops.

$$P_{\text{OOD}}(y_0) = P_{\text{OOD}}(y_1) \implies \text{Accuracy}_{\text{OOD}}(f) < \text{Accuracy}_{\text{IID}}(f) \tag{2.10}$$

2.1.4 Attribute Generalization

Attribute generalization is treated as a separate category by the previous literature (Idrissi et al., 2022; Yang et al., 2023), although, in essence, it is an extreme case of attribute imbalance, where examples with certain attributes are missing in the training set, but are available in the test set.

$$P_{\text{IID}}(a_u) = 0 \text{ and } P_{\text{OOD}}(a_u) > 0 \tag{2.11}$$

Under this setting, models can have lower confidence on examples with unseen attributes.

$$P_f(y|a_u, x) < P_f(y|x) \tag{2.12}$$

In OOD test sets, if the examples with a_u that the model is less confident about are more frequent, this can lead to overall performance drops.

$$P_{\text{OOD}}(a_u) > 0 \implies \text{Accuracy}_{\text{OOD}}(f) < \text{Accuracy}_{\text{IID}}(f) \tag{2.13}$$

2.2 Loss-Function-Based Approaches

2.2.1 Empirical Risk Minimization

Consider a setting where $x \in X$ are the input features and $y \in Y$ are labels. Empirical Risk Minimization (ERM) (Vapnik, 1999), aims to find a model f_θ, parameterized by θ, that minimizes the expected loss $L(f_\theta(x), y)$ under the training distribution, P_{IID}.

$$\theta_{ERM} = \arg\min_{\theta} \mathbb{E}_{(x,y)\sim P_{IID}}[L(f_\theta(x), y)] \tag{2.14}$$

Since every (x, y) in a batch are randomly sampled from P_{train} during training, the contribution of each example in a batch to the loss function is weighted equally. Let X_m be the input features for the examples in a batch of size m and Y_m be the corresponding labels. The value of the loss function is the average of the losses for each example.

$$L(f_\theta(X_m), Y_m) = \frac{1}{m}\sum_{i=1}^{m} L(f_\theta(x_i), y_i), \tag{2.15}$$

where (x_i, y_i) are the samples in the current batch. The gradient of the batch is the average of the gradients of the loss function with respect to each example.

$$\nabla_\theta L(f_\theta(X_m), Y_m) = \nabla_\theta \frac{1}{m}\sum_{i=1}^{m} L(f_\theta(x_i), y_i) = \frac{1}{m}\sum_{i=1}^{m} \nabla_\theta L(f_\theta(x_i), y_i), \tag{2.16}$$

where ∇_θ denotes the gradient with respect to θ. Therefore, every example in the batch contribute equally to learning via the gradient updates.

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta L(f_\theta(X_m), Y_m), \tag{2.17}$$

where η is the learning rate. This setup allows us to find models that achieve high test accuracy on identically distributed test sets by minimizing the expected loss under P_{IID}. However, models trained with ERM often struggle with predictions on examples from data groups that are underrepresented in P_{IID} (Blodgett et al., 2016; Duchi et al., 2022; Hashimoto

et al., 2018; Hovy and Søgaard, 2015; Tatman, 2017; Yang et al., 2023). This is not surprising since each group contributes to the gradient updates proportional to their representation in the training set. Hence, over-represented groups end up being more influential in learning. As a result, changes in the proportion of data groups in test distribution, P_{OOD}, results in drops in the model performance.

2.2.2 Modifying the Loss Function

The methods that aim to make the performance of the models more robust to distribution shifts by ensuring consistent performance across different data groups are often grouped into two main categories: (1) the methods that use the knowledge of data groups during the training and (2) the methods that do not require knowledge of the data groups. The two categories are inherently connected with an assumption that enables the methods in the second category to improve model performance on underrepresented data groups in the absence of information on data groups during training: If we up-weight the contribution of difficult examples to the loss function and down-weight the contribution of easy examples, we can learn models that perform better on more difficult examples. Here, we use the terms "easy" and "difficult" to refer to the difficulty from the perspective of the model. For example, if a model assigns a very high probability to the correct label for an example, we can call that example easy. Below, we provide a more detailed explanation of what we mean by "easy" and "difficult" examples and offer an intuitive reasoning for why these methods work.

Difficult Examples

The examples for which the model has low confidence (low probability assigned to the true label) are difficult examples. The reason behind this low confidence can be:

1. **Spurious Correlations (Minority Attribute-Label Pair):** The example has an attribute-label pair that contradicts a spurious correlation that is predictive for the majority of the examples in the training set. The impact of the shortcut rule leads to low confidence for the example.
2. **Attribute Imbalance (Minority Attribute):** The example contains a pattern that is rare in the training set.
3. **Label Imbalance (Minority Label):** The example has a label that is rare in the training set.
4. **Other:** The example has noise in the input or the label; the example is ambiguous, etc.

Intuition for Up-weighting Difficult Examples

A loss function that up-weights the contribution of examples with attributes, labels, and attribute-label pairs that are underrepresented in the training set incentivizes the model to learn a classification rule that works well for examples from different data groups.

Easy Examples

The examples for which the model has high confidence (high probability assigned to the true label) are easy examples. The reason behind this high confidence can be:

1. **Spurious Correlations (Majority Attribute-Label Pair):** This is the "too good to be true" scenario. The example has an attribute-label pair that aligns with a spurious correlation that is predictive for the majority of the examples in the training set. The model implements a shortcut rule that relies on this spurious correlation and predicts the label with high confidence.
2. **Attribute Imbalance (Majority Attribute):** The example contains a pattern that dominates the training set.
3. **Label Imbalance (Majority Label):** The example has a label that dominates the training set.
4. **Other:** The example follows a simple pattern, belongs to a class that is well-separated, etc.

Intuition for Down-weighting Easy Examples

A loss function that down-weights the contribution of examples with attributes, labels, and attribute-label pairs that are overrepresented in the training set incentivizes the model to learn a classification rule that does not rely on spurious correlations, is not biased towards certain labels, and has similar confidence for examples with different attributes. Moreover, when easy examples are a substantial majority in the training set, their aggregate influence on the loss function can be significant even when they are classified correctly, which can prevent the fine-grained gradient updates that are required to find parameters that correctly classify more difficult examples (Lin et al., 2018). Hence, down-weighting their contribution can promote learning the rules that apply to the minority examples.

Different Modifications to the Loss Function

Various approaches have been proposed to adjust the contribution of examples to learning. These include Invariant Risk Minimization (IRM) (Arjovsky et al., 2020) which incentivizes the model to focus on causal relationships, Conditional Value at Risk DRO (CVaRDRO) (Duchi and Namkoong, 2020) which optimizes for the worst group performance within a specified risk level, Mixup (Zhang et al., 2018), which uses data augmentation to favor simple linear behavior between examples, Deep Feature Reweighting (DFR) (Kirichenko et al., 2023), which re-trains the last layer of the network on a group balanced sample of examples, among others Cao et al. (2019); Cui et al. (2019); Kang et al. (2020); Ren et al. (2020). In the next sections, we focus on 4 different approaches to weighting examples, which are described in Table 2.1. These include one method that assumes the knowledge of data groups in training (GroupDRO) and three methods that do not require the knowledge of the data groups: Focal Loss trains a single network, Just-Train Twice trains two networks sequentially, and Learning From Failure trains two networks simultaneously. We selected these four approaches because they cover a variety of different techniques used in the literature.

Method	**Procedure**
Group DRO	Weights the examples with the difficulty scores assigned to the groups based on the per group loss
Focal Loss	Down-weights examples that are predicted with high confidence
JTT	Up-weights examples that are misclassified by ERM
LfF	Weights the examples with difficulty scores assigned to each example based on the confidence of a second bias model for the example

Table 2.1 Modifications to the Loss Function

2.2.3 Distributionally Robust Optimization

Group Distributionally Robust Optimization (GroupDRO) (Duchi et al., 2018; Sagawa et al., 2020) is a learning algorithm that assumes the knowledge of data groups during training. It modifies the loss function to find the parameters θ that minimize the expected loss for the group that has the maximum expected loss. We denote the data groups with $g \in G$.

$$\theta_{DRO} = \arg\min_{\theta} \left\{ \max_{g \in G} \mathbb{E}_{(x,y)\sim P_{IID|g}} [L(f_\theta(x), y)] \right\} \tag{2.18}$$

To estimate θ_{DRO}, Sagawa et al. (2020) has proposed an algorithm (Sagawa et al., 2020, Online Optimization Algorithm for group DRO) that assigns an importance weight, q_g, to each group, g, and dynamically updates these importance weights based on the group's loss. The data groups are assigned equal importance at the beginning of training. In each gradient update step, the algorithm randomly selects one of the data groups, g, samples examples from g, and computes the loss for these examples.

$$L_g = \sum_{(x_i,y_i)\in g} L(f_\theta(x_i), y_i) \tag{2.19}$$

Then, the algorithm updates the importance of the group.

$$q_g \leftarrow q_g \cdot \exp(\eta_{DRO} \cdot L_g), \tag{2.20}$$

where η_{DRO} is a hyperparameter. After the group weights are normalize to add up to 1, the algorithm updates the model weights.

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta q_g L_g \tag{2.21}$$

As a result, GroupDRO up-weights the contribution of examples from difficult groups to learning.

2.2.4 Focal Loss

Focal Loss (Lin et al., 2018) modifies the loss function to down-weight the contribution of the examples that are predicted with high confidence to the loss function. The method does not assume the knowledge of the data groups during training and is initially designed to address the problem of imbalance between foreground and background labels in object detection for image processing (Lin et al., 2018). Remember our loss function in Empirical Risk Minimization.

$$L(f_\theta(X_m), Y_m) = \frac{1}{m}\sum_{i=1}^{m} L(f_\theta(x_i), y_i) \tag{2.22}$$

Focal Loss rewieghts the contribution of examples to the loss-function-based on the confidence of the model for

$$L_{\text{Focal}}(f_\theta(X_m), Y_m) = \frac{1}{m}\sum_{i=1}^{m} (1-p_i)^\gamma L(f_\theta(x_i), y_i), \tag{2.23}$$

where p_i is the probability that the model assigns to the true label for example i, and γ is the focusing hyperparameter. For each example, p_i is calculated from the logits vector z, where each element z_j is the logit that the model assigns to a particular label and z_i is the logit that corresponds to the true label: $p_i = \frac{e^{z_i}}{\sum_{j=1}^{N} e^{z_j}}$ The learning algorithm selects the parameters that minimize the expected Focal Loss. As a result, Focal Loss down-weights the influence of "easy examples" that are predicted with high probability on the gradient updates.

2.2.5 Just Train Twice

Just-Train-Twice (JTT) (Liu et al., 2021) up-weights the examples that are misclassified by an ERM-trained model. The method does not assume the knowledge of the data groups during training. Training epochs are divided into two stages. In the first stage, the learning algorithm first trains a model with ERM for several epochs. Then, it identifies the set, E, of examples in the training set that are misclassified by this model.

$$E = \{x_i \in X \text{ such that } f_{\theta_{STAGE1}}(x_i) \neq y_i\}$$

In the second stage, JTT trains a second model with a modified loss function L_{JTT} that up-weights the influence of the examples that are misclassified by the first stage model.

$$L_{\text{JTT}}(f_\theta(X_m), Y_m) = \left(\lambda_{JTT} \sum_{x_i \in E} L(f_\theta(x_i), y_i) + \sum_{x_i \notin E} L(f_\theta(x_i), y_i) \right),$$

where λ_{JTT} is a hyperparameter. As a result, the influence of the examples that are misclassified by the first stage model is amplified λ_{JTT} times in the second stage.

2.2.6 Learning From Failure

Learning from Failure (LFF) (Nam et al., 2020) simultaneously trains two models: A bias model that is used to assign difficulty scores to examples and a debiased model whose loss function up-weights the examples with high difficulty scores. The biased model has a loss function that resembles the Focal Loss. However, instead of up-weighting the difficult examples, it up-weights the easy examples.[1] The loss function of the biased model, f_{θ_B}, is

$$L_{\text{Bias}}(f_{\theta_B}(x_i), y_i) = \mathfrak{q} p_i^{\mathfrak{q}} L(f_{\theta_B}(x_i), y_i), \tag{2.24}$$

[1] Our description of biased loss follows the implementation from the codebase of the authors (Nam et al., 2020). The generalized cross entropy loss formula from the paper is $L_{\text{Bias}}(f_\theta(x_i), y_i) = \frac{(1-p_i)^{\mathfrak{q}}}{\mathfrak{q}} L(f_\theta(x_i), y_i)$.

where $\mathfrak{q}$ is a hyperparameter, and the loss function for the debiased model, f_{θ_D}, is

$$L_{\text{Debias}}(f_{\theta_D}(x_i), y_i) = W(x_i) L(f_{\theta_D}(x_i), y_i), \tag{2.25}$$

where the difficulty score for the example, $W(x_i)$, is computed as

$$W(x_i) = \frac{L(f_{\theta_B}(x_i), y)}{L(f_{\theta_B}(x_i), y) + L(f_{\theta_D}(x_i), y)}, \tag{2.26}$$

As a result, LFF down-weights the examples for which the biased model has high confidence (easy examples) and up-weights the examples for which the biased model has low confidence.

Chapter 3

Methodology

The loss-function-based approaches that we discussed in the previous chapter are model agnostic: They do not make assumptions about the functional form of the model f, treating it as a black box function. Motivated by the prominence of Transformers as the state-of-the-art models for text classification, in this chapter, we focus on the specific setting where f is a Transformer model. In Section 3.1, we explain the internal processes of Transformer models that compute the internal representations. In Section 3.2, we introduce our Competing Rules Hypothesis (CRH) and discuss two lines of suggestive evidence that motivate our formulation. Finally, in Section 3.3, we present REPRSHIFT.

3.1 The Residual Stream Perspective (Overview)

We use the idea of the "residual stream" Elhage et al. (2021) to understand the computations of this Transformer model. This framework interprets Attention and MLP layers as contributors to the residual stream, which is the central object of the model. Attention and MLP layers take a checkpoint of the residual stream as input, make a computation, and add the result of their computation back to the residual stream. In other words, they *"read information from"* and *"write information to"* the residual stream (Elhage et al., 2021).

3.1.1 A Transformer Layer

Figure 3.1 demonstrates a Transformer layer from the BERT model (Devlin et al., 2019).[1] We call the input of the Transformer layer, r^{ℓ}. This is the checkpoint of the residual stream that our Transformer layer takes as input. r^{ℓ} contains vectors of size d_{model} for each token

[1] We use bert-base-uncased in our experiments. Therefore, we focus our discussion on BERT's computations.

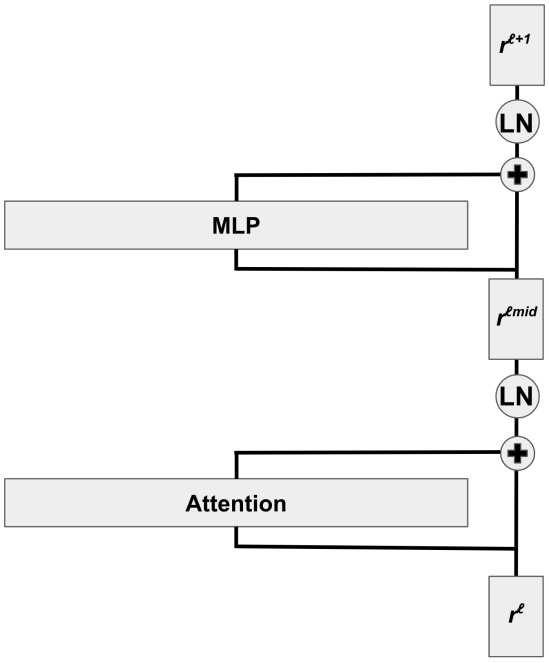


Fig. 3.1 Transformer Layer. The MLP and Attention Layers are illustrated as contributors to the residual stream. LN represents layer normalization.

position in the context.[2] Hence, we can think of r^{ℓ} as a matrix with d_{context} rows and d_{model} columns. We define a second checkpoint of the residual stream, $r^{\ell\text{mid}}$, after the attention layer and the first layer normalization.

$$r^{\ell\text{mid}} = \text{LayerNorm}_1(r^{\ell} + \text{Attn}(r^{\ell})), \tag{3.1}$$

where Attn(.) denotes an encoder (bidirectional) attention layer (Devlin et al., 2019; Vaswani et al., 2023) and LayerNorm(.) denotes layer normalization (Ba et al., 2016). The third checkpoint of the residual stream is the output of the Transformer layer, $r^{\ell+1}$.

$$r^{\ell+1} = \text{LayerNorm}_2(r^{\ell\text{mid}} + \text{MLP}(r^{\ell\text{mid}})), \tag{3.2}$$

where MLP(.) denotes an MLP layer (Vaswani et al., 2023). Note that all the checkpoints of the residual stream, r^{ℓ}, $r^{\ell\text{mid}}$, and $r^{\ell+1}$, have the same dimensions. Attn(.) and MLP(.) take a checkpoint as input, make computations to compute a value, and add that value back to the residual stream.

[2] $d_{\text{model}} = 768$ for bert-base-uncased.

3.1.2 From Text to Logits

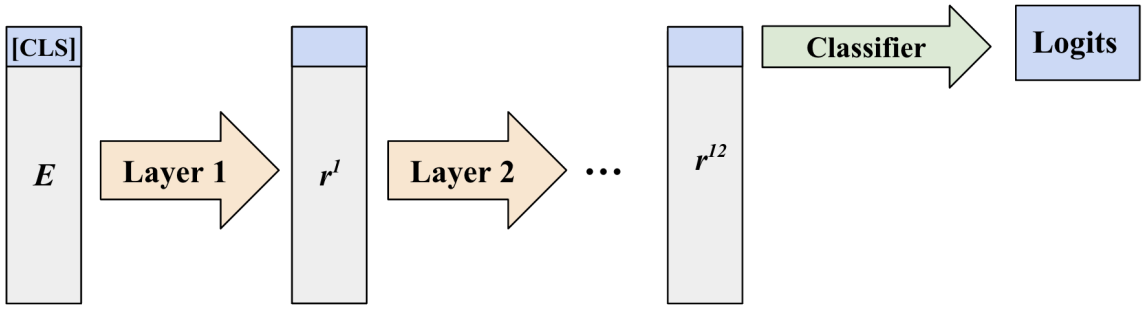


Fig. 3.2 Model Computations from Tokens to Logits

From Text to Tokens

Figure 3.2 demonstrates how an input x_i is passed through the model to compute the logits. Note that our text classification model takes a text as input. Firstly, the text inputs are tokenized using WordPiece tokenization (Devlin et al., 2019; Wu et al., 2016) with a vocabulary size of d_{vocab}.[3] We follow the tokenization convention from Devlin et al. (2019), prepending a special [CLS] token and appending a special [SEP] token to all sequences. For our single sentence classification task, toxicity detection, this results in "[CLS] + Comment + [SEP]". For NLI, we add an additional [SEP] token between the premise and hypothesis, resulting in "[CLS] + Premise + [SEP] + Hypothesis + [SEP]" following Devlin et al. (2019). For sequences that are longer than d_{context}, we discard tokens so that the length of the sequence equals d_{context}.[4] The sequences that are shorter than d_{context} are padded by appending the special [PAD] token (Devlin et al., 2019). As a result, we obtain a vector of size d_{context} that represents a single example, where the entries of the vector contain the vocabulary indices for each token in the example.

From Tokens to Embeddings

Each of the d_{context} tokens in an example is represented by 3 vectors of size d_{model}. Firstly, each index in the vocabulary has a corresponding d_{model} dimensional **token embedding** that is learned during training. Secondly, each position in the context has a corresponding d_{model} dimensional **positional encoding** (Vaswani et al., 2023). Thirdly, to indicate whether the

[3] We use bert-base-uncased tokenizer with $d_{\text{vocab}} = 30,000$ in our experiments.

[4] We set a fixed context length of $d_{\text{context}} = 128$ for MultiNLI and $d_{\text{context}} = 220$ for CivilComments following Idrissi et al. (2022) and Yang et al. (2023). When discarding tokens from sequences that are longer than $d_{\text{context}} = 128$ for MultiNLI, we iteratively discard tokens from the longer one of the Premise and Hypothesis.

token belongs to the first or the second sequence (premise or hypothesis in NLI), the model uses two d_{model} dimensional **token type identifiers** (Devlin et al., 2019). The embedding for each position in the context is then calculated as the sum of the token embedding, the positional encoding, and the token type identifier. This value is normalized by passing it through a LayerNorm(.) (Ba et al., 2016) to obtain the d_{model} dimensional embedding e for a given position. For input x, we represent the embeddings for all positions as the matrix $E \in \mathbb{R}^{d_{context} \times d_{model}}$, where each row of E contains an embedding e that corresponds to a position in the context.

From Embeddings to Representations

The embedding matrix E is the first checkpoint of the residual stream, r^0. $E = r^0$ is passed through the first Transformer layer, described in Section 3.1, to obtain r^1. As shown in Figure 3.2, for the BERT-base model, which has 12 layers, this process is repeated for 12 Transformer layers to obtain the final representation of the input r^{12}. This matrix, $r^{12} \in \mathbb{R}^{d_{context} \times d_{model}}$, has the same dimensions as E. The rows of r^{12} correspond to the positions of each of the $d_{context}$ tokens in the context. Among these rows, only the first row of r^{12} that corresponds to the "[CLS]" token position in E is used by the classifier to compute the logits. Therefore, after the input is passed through 12 Transformer layers, it is represented as a single d_{model} dimensional vector, which we call r^{12}_{CLS}. More generally, the first row of r^{ℓ} that corresponds to the position of the CLS token is a d_{model} dimensional vector, which we call r^{ℓ}_{CLS}.

From Representations to Logits

Finally, the classifier of our model takes r^{12}_{CLS} as input and passes it through a 2-layer MLP with a tanh activation function in the middle. The classifier computations can be represented with matrix multiplication.

$$\text{Classifier}(r_i) = \tanh(r_i \cdot W_P + b_P) \cdot W_U + b_U, \tag{3.3}$$

where $r_i \in \mathbb{R}^{d_{model}}$ is the input, $W_P \in \mathbb{R}^{d_{model} \times d_{model}}$ contains the weights of the first MLP layer with bias term $b_P \in \mathbb{R}^{d_{model}}$ and $W_U \in \mathbb{R}^{d_{model} \times d_{labels}}$ contains the weights of the second MLP layer with bias term $b_U \in \mathbb{R}^{d_{labels}}$. This results in a d_{labels} dimensional Logits vector, where each element corresponds to the logits for a certain label.

$$\text{Logits} = \text{Classifier}(r^{12}_{\text{CLS}}) \tag{3.4}$$

The model selects the predicted label to be the label with the highest corresponding logit.

3.2 Competing Rules Hypothesis (CRH)

In this section, we present Competing Rules Hypothesis (CRH) as a toy model of shortcut learning. We present two lines of supporting evidence based on our observations of an ERM-trained model's behavior on our natural language inference task. A third line of evidence is presented in our experiments in Section 4.4.1.

3.2.1 Formulation

CRH describes the model's decision-making process as a competition between an intended rule and a shortcut rule. The shortcut rule exploits spurious correlations to boost the performance of the model for majority groups, which hurts the performance of minority groups. The intended rule implements an algorithm that generalizes across different data groups. Algorithms 1 and 2, which we have presented in Section 1.4, describe the shortcut rule and intended rule implemented by a model trained on our NLI dataset. We define the shortcut rule and the intended rule more generally in Algorithms 4 and 5. We hypothesize that the

Algorithm 4 Shortcut Rule (general)

1: **if** x contains attribute a_{shortcut} **then**
2: Influence the logits to predict $P_{IID}(y|a_{\text{shortcut}})$
3: **end if**

Algorithm 5 Intended Rule (general)

1: Influence the logits to predict $P_{IID}(y|x_R)$

predictions of a model that has learned a shortcut are determined by a competition between the shortcut rule and the intended rule. This competition takes place in the logit space, the d_{labels} dimensional output of the model, where each dimension determines the model's preference for one of the labels. In general, the shortcut rule influences the logits to predict $P_{IID}(y|a_{\text{shortcut}})$, where a_{shortcut} is a spuriously correlated attribute, which we call shortcut attribute. P_{IID} is the data distribution from which the training set is sampled. For NLI, the shortcut rule increases the logit corresponding to the contradiction label whenever there is a negation word in the hypothesis. Meanwhile, the intended rule influences the logits to predict $P_{IID}(y|x_R)$, where x_R are the robust features that generalize across the data groups. In NLI, this accounts for the logical relationship between the two sentences. The model predicts the label that has the highest logit. We note that the core idea behind our formulation resembles a framework developed by Ortu et al. (2024) to study a different problem, in-context learning

and the treatment of factual knowledge by LLMs. Their work inspired our thinking. We provide two lines of suggestive evidence for CRH in the next two sections. Section 4.4.1 presents a third line of evidence.

3.2.2 Suggestive Evidence for CRH

	No Shortcut Attribute			Shortcut Attribute		
	Contradiction	Entailment	Neutral	Contradiction	Entailment	Neutral
Predicted Contradiction	79.9	4.3	11.4	**96.7**	14.1	27.8
Predicted Entailment	6.5	86.1	11.2	1.1	75.1	7.7
Predicted Neutral	13.6	9.7	77.4	2.3	10.9	64.5

Table 3.1 MultiNLI Per Attribute Confusion Matrix (Validation). 6 columns indicate the 6 data groups, and the rows are predicted labels. The values inside the cells are the percentage of examples that are predicted as contradiction, entailment, or neutral for each group. Hence, the column sums add up to 100%. The reported results are the averages on the validation set from 3 fine-tuning runs. ERM is used as the fine-tuning loss function.

Suggestive Evidence 1: Constructive Interference

If CRH is true, on average, we would expect to see especially high accuracy for examples that have (i) true contradiction labels and (ii) have shortcut attributes (negation word in the hypothesis). In that case, both the intended and the shortcut rule would influence the prediction towards contradiction, and, instead of competing, the two rules *join forces* to predict contradiction. This can be observed in the first row on the fourth column in Table 3.1. In particular, the examples that have true contradiction labels are predicted to be contradiction on average 79.9% of the time in the absence of the shortcut attributes, whereas, in the presence of the shortcut attributes, the likelihood increases to 96.7%. We call this *constructive interference* between the shortcut rule and the intended rule. When we take a step inside the model, we observe similar patterns in the logit space. In Table 3.2, we see especially high logits (4.03) for contradiction in examples that have true contradiction labels and shortcut attributes.

Suggestive Evidence 2: Destructive Interference

If CRH is true, on average, we would expect our model to predict the examples that contain shortcut attributes to be contradiction more often regardless of their true label. This phenomenon can be observed in the first row of Table 3.1. In particular, we observe an increase

in the probability of being predicted as a contradiction for the examples with entailment (4.3% → 14.1%) and neutral (11.4% → 27.8%) labels. In this case, the two rules *oppose* each other, and the *destructive interference* between the shortcut rule and intended rule leads to lower accuracies for labels entailment (86.1 → 75.1%) and neutral (77.4 → 64.5%) in the presence of shortcut attributes. We note that destructive interference seems to be the reason for poor performance in worst-performing groups. This observation directly translates to the logit space: In Table 3.2, we observe that the average contradiction logits are higher in the presence of the shortcut attribute regardless of the true label of the example. On average, the presence of a shortcut attribute increases the contradiction logits by 1.0 logit units and decreases the logits that correspond to the entailment and neutral labels around half logit units (see Table A.1 for a detailed breakdown).

	No Shortcut Attribute			Shortcut Attribute		
	Contradiction	Entailment	Neutral	Contradiction	Entailment	Neutral
Contradiction Logit	2.91	-1.94	-1.20	**4.03**	-1.29	0.02
Entailment Logit	-2.26	2.79	-1.26	-3.02	2.20	-1.80
Neutral Logit	-0.90	-0.65	2.52	-1.34	-0.63	1.75

Table 3.2 Average Logits per Data Group. Each column demonstrates the average logits given to each label for the corresponding data group for examples from Table 3.1.

3.3 Representation Shift (REPRSHIFT)

In this section, we build on CRH to develop an interpretability-based approach to shortcut mitigation. Previously, we observed that the shortcut rule and the intended rule seem to be competing in the logit space to influence the ERM-trained model's predictions. Based on this observation, we consider *suppressing the shortcut rule* to allow the intended rule to dominate the predictions and improve the generalization of the ERM-trained model across data groups.

3.3.1 Model Algebra

To suppress the shortcut rule, we add an "Inverse Shortcut" rule into the ERM-trained model. Assuming we can do simple *model algebra*, our reasoning steps are:

We want to have a model with only INTENDED RULE

We have a model with INTENDED RULE + SHORTCUT RULE

We compute INVERSE SHORTCUT RULE = − SHORTCUT RULE

We add inverse shortcut rule into our model and get a model with
INTENDED RULE + SHORTCUT RULE + INVERSE SHORTCUT RULE

This is equal to
INTENDED RULE + SHORTCUT RULE + (− SHORTCUT RULE)

Shortcut rules cancel out and we get a model with
INTENDED RULE

We expect this procedure to eliminate destructive interference and improve the accuracy of the worst group. We define the inverse shortcut rule in Algorithm 6. We call it the "inverse" shortcut rule, because it is the additive inverse of the shortcut rule in our *model algebra*.

Algorithm 6 Inverse Shortcut Rule (general)

1: **if** x contains attribute a_{shortcut} **then**
2: Influence the logits to predict $(1 - P_{IID}(y|a_{\text{shortcut}}))$
3: **end if**

Related Work

Previous works in model editing have demonstrated that it is possible to add objects to or remove objects from generated images (Bau et al., 2020) and edit factual knowledge in LLMs (De Cao et al., 2021; Meng et al., 2023a; Mitchell et al., 2022a,b). The methods from previous studies include constrained fine-tuning (Zhu et al., 2020), training hyper-networks to predict the necessary weight updates (De Cao et al., 2021; Hase et al., 2021; Mitchell et al., 2022a; Sinitsin et al., 2020), and retrieval augmented model edits which does not change the weights of the original model (Mitchell et al., 2022b). Recently, a new approach, rank-one model editing, has been proposed by (Meng et al., 2023a), which can surgically modify a single layer in the network to inject a new fact into an LLM. Moreover, Meng et al. (2023b) built on this framework to inject multiple facts into the same model via model edits. After we compute the inverse shortcut rule, we use the rank-one model editing approach from Meng et al. (2023a) to inject the inverse shortcut rule into the model.

3.3.2 REPRSHIFT

REPRSHIFT, modifies a single layer inside the network to shift the representations of examples with shortcuts. We implement REPRSHIFT in 4 steps:

1. We compute an internal representation for a_{shortcut}, which we call k_{shortcut}, to implement the **if** statement in the inverse shortcut rule (Algorithm 6, Line 1).

2. We compute a logit shift vector, $v_{\text{logitshift}}$, that quantifies the impact of the shortcut rule on the logits, $P_{IID}(y|a_{\text{shortcut}})$.

3. We use this logit shift vector to compute a representation shift vector, $v_{\text{reprshift}}$, that implements the **then** statement in the inverse shortcut rule (Algorithm 6, Line 2).

4. Finally, we modify a single MLP weight matrix W_{out} inside our model to implement the inverse shortcut rule:

$$\textbf{if } k_{\text{shortcut}}, \textbf{ then } v_{\text{reprshift}} \quad \text{or, equivalently,} \quad k_{\text{shortcut}} \cdot W_{out} = v'_{\text{reprshift}},$$

   where $v'_{\text{reprshift}}$ is a small adjustment to $v_{\text{reprshift}}$, which we discuss in Equation 3.3.2.

In Step 4, to modify W_{out}, we use the model editing approach introduced by Bau et al. (2020) with the closed form solution derived by Meng et al. (2023a). The following sections address each of the four steps. We use the letter ℓ_{edit} to denote the layer of the model we are modifying. In our experiments, we set $\ell_{\text{edit}} = 12$ and modify the MLP layer in the last (12th) Transformer layer of our model. Section 4.4.1 discusses the reasoning behind this design decision. Appendix A.2 contains the implementation details for our experiments.

Anatomy of an MLP Layer

The MLP layer depicted in Figure 3.1 consists of two fully connected layers. The first fully connected layer maps the input from d_{model} to a higher dimension, d_{mlp}.[5] This value is passed through a Gaussian Error Linear Unit (GELU) activation function (Hendrycks and Gimpel, 2023). Then, a second fully-connected layer maps this value back to d_{model}.

$$\text{MLP}(r) = \text{GELU}(r \cdot W_{in} + b_{in}) \cdot W_{out} + b_{out}, \tag{3.5}$$

where $r \in \mathbb{R}^{d_{context} \times d_{model}}$ is the checkpoint of the residual stream that the MLP layer takes as input. $W_{in} \in \mathbb{R}^{d_{model} \times d_{mlp}}$ contains the weights of the first fully-connected layer with bias term $b_{in} \in \mathbb{R}^{d_{mlp}}$. $W_{out} \in \mathbb{R}^{d_{mlp} \times d_{model}}$ contains the weights of the second fully-connected layer with bias term $b_{in} \in \mathbb{R}^{d_{model}}$.

[5] In our experiments, $d_{model} = 768$ and $d_{mlp} = 3072$ (Devlin et al., 2019).

Step 1: Internal Representation for the Shortcut Attribute

To compute an internal representation for the shortcut attribute, we get a label-balanced sample of examples that contain shortcut attributes from the validation set. We run these examples through the model and save the representations generated inside the MLP in layer ℓ_{edit} after the GELU activation.

$$\text{S} = \text{GELU}(r^{(\ell_{\text{edit}}-1)mid} \cdot W_{in} + b_{in}) \tag{3.6}$$

See Equation 3.1.1 for a definition of $r^{\ell mid}$. Note that for a given input, this results in a signal matrix $S \in \mathbb{R}^{d_{context} \times d_{mlp}}$. We take only the first row of this matrix that corresponds to the [CLS] token position, which gives us a d_{mlp} dimensional vector, s_{CLS}, for each example. To obtain $k_{shortcut}$, we take the average of s_{CLS} across all examples. By averaging representations across a label-balanced subset of examples with shortcut attributes, we aim to eliminate the variation in the representations due to factors other than the shortcut attribute. We call $k_{shortcut} \in \mathbb{R}^{d}_{mlp}$ the internal representation of the shortcut attribute. We note that a similar averaging approach has also been used by Meng et al. (2023a) to compute a representation for a "subject token" (Meng et al., 2023a, Equation 3).

Step 2: Logit Shift Vector

We use the same label-balanced sample of examples that contain shortcut attributes from the validation set to compute the logit shift vector. $v_{\text{logitshift}} \in \mathbb{R}^{d_{\text{labels}}}$ is a vector that is added to the logits before the label with the maximum logit is selected to be the predicted label. On our sample from the validation set, we grid search for $v_{\text{logitshift}}$ that maximizes the worst group accuracy on the validation set. As a result, when added to the logits, $v_{\text{logitshift}}$ leads to consistent performance across labels in examples that contain shortcut attributes. Implementation details can be found in Appendix A.2.2.

Step 3: Representation Shift Vector

In this step, in addition to our label-balanced sample of examples that contain shortcut attributes, we take another label-balanced sample of examples that do not contain shortcut attributes. The union of two samples contains a group-balanced sample from the validation set. Then, we search for $v_{\text{reprshift}}$, such that if the MLP in layer ℓ_{edit} outputs $v_{\text{reprshift}}$ for the position that corresponds to the [CLS] token, the logits are shifted by $v_{\text{logitshift}}$. To compute

$v_{reprshift}$, we set up this objective as an optimization problem.

$$L_{reprshift}(x_i, v_{reprshift}) = \|f^{\text{logits}}_{v_{reprshift}}(x_i) - \left[f^{\text{logits}}(x_i) + v_{logitshift}\right]\|^2, \tag{3.7}$$

where $L(x_i, v_{reprshift})$ is the loss function, $f^{\text{logits}}_{v_{reprshift}}(x_i)$ is logit output of our model for x_i, where the output of the MLP in layer ℓ_{edit} is set to $v_{\text{reprshift}}$. $f^{\text{logits}}(x_i)$ is the logit output of our initial model. In principle, the vector $v_{\text{reprshift}}$ that minimizes this loss function would shift the logits by $v_{\text{logitshift}}$ whenever it is outputted by the MLP in layer ℓ_{edit}. To minimize the impact of the representation shift on examples that do not have shortcut attributes, we also use a control loss function:

$$L_{control}(x_i, v_{reprshift}) = \|f^{\text{logits}}_{v_{reprshift}}(x_i) - f^{\text{logits}}(x_i)\|^2, \tag{3.8}$$

We select $v_{reprshift}$ by simultaneously minimizing $L_{reprshift}$ for examples with shortcut attributes and $L_{control}$ for examples with no shortcut attributes.

$$\min_{v_{reprshift}} \sum_{x_i} \begin{cases} L_{reprshift}(x_i, v_{reprshift}) & \text{if } x_i \text{ has a shortcut attribute,} \\ L_{control}(x_i, v_{reprshift}) & \text{if } x_i \text{ has no shortcut attribute.} \end{cases} \tag{3.9}$$

Note that all of the parameters of our model are fixed, and the only changing model component is the output of the MLP in layer ℓ_{edit}, which we set to $v_{reprshift}$. We select $v_{reprshift}$ by minimizing Equation 3.3.2 via gradient descent. Implementation details are included in Appendix A.2.3.

Step 4: Modifying Model Weights

Finally, we modify the weight matrix W_{out} (see Equation 3.3.2) within the MLP in layer ℓ to implement the Reverse Shortcut Rule: Whenever W_{out} takes a shortcut signal, k_{shortcut}, as input, we want the output of the MLP to be $v_{\text{reprshift}}$, so that the logits are shifted by $v_{\text{logitshift}}$. Remember the functional form of the MLP layer, which we previously defined in Section 3.3.2.

$$\text{MLP}(r) = \text{GELU}(r \cdot W_{in} + b_{in}) \cdot W_{out} + b_{out}, \tag{3.10}$$

Note that the shortcut attribute representation, k_{shortcut}, is calculated as the output of the GELU. Hence, to implement the reverse shortcut rule, we need a W_{out} such that,

$$v_{\text{reprshift}} = k_{\text{shortcut}} \cdot W_{out} + b_{out} \tag{3.11}$$

Rearragning the terms, we get

$$v'_{\text{reprshift}} = k_{\text{shortcut}} \cdot W_{out}, \tag{3.12}$$

where $v'_{\text{reprshift}} = v_{\text{reprshift}} - b_{out}$. We want to insert this association between k_{shortcut} and $v_{\text{reprshift}}$ into W_{out} optimally in a way that preserves the output of W_{out} for any other input k when $k \neq k_{\text{shortcut}}$. This problem has been previously studied by Meng et al. (2023a) and Bau et al. (2020), who interpret the weights of a fully-connected layer, W, as linear associative memory that assigns keys $k_1, k_2, \ldots, k_n$ to values $v_1, v_2, \ldots, v_n$ via matrix multiplication, such that $Wk_i = v_i$ (Anderson, 1972; Kohonen, 1972). W takes keys k_i as input and outputs the corresponding values v_i. For a more compact notation, we can stack the inputs in a keys matrix K, where $K = [k_1|k_2, \ldots|k_n]$, and do the same for the values to obtain $V = [v_1|v_2|\ldots|v_n]$. W can map the keys in K to the corresponding values in V by solving $WK \approx V$ (Bau et al., 2020; Meng et al., 2023a). Under this setting, the optimum W can be calculated by solving $W = VK^+$, where K^+ is the pseudo-inverse of K (Bau et al., 2020; Meng et al., 2023a). Bau et al. (2020) has demonstrated that a new key-value pair $(k_{\text{new}}, v_{\text{new}})$ can be inserted into W with a minimum effect on the other key-value pairs by solving the following optimization problem.

$$\min_W \|WK - V\|^2 \quad \text{such that} \quad Wk_{\text{new}} = v_{\text{new}}, \tag{3.13}$$

Moreover, Meng et al. (2023a) has derived a closed-form solution for this least-squared optimization problem.

$$W^{\text{new}} = W + \Lambda(C^{-1}k_{\text{new}})^T, \tag{3.14}$$

where W^{new} is the new weights matrix that also contains the association between k_{new} and v_{new}, $C = KK^T$, and $\Lambda = (v_{\text{new}} - Wk_{\text{new}})/(C^{-1}k_{\text{new}})^T k_{\text{new}}$. For a full proof, we refer the reader to Meng et al. (2023a, Appendix A). We use the same approach to insert the association between k_{shortcut} and $v'_{\text{reprshift}}$ into W_{out}. To compute the columns k_i of the keys matrix K, we save the internal representations, s_{CLS} (defined in Section 3.3.2) from a group balanced sample from the validation set (See Appendix A.2.4 for implementation details). To obtain the columns v_i of the values matrix V, we compute $v_i = k_i \cdot W_{out}$ for each k_i. Finally, we modify W_{out} to insert the new key-value pair $(k_{\text{shortcut}}, v'_{\text{reprshift}})$ as follows.

$$W_{out}^{\text{new}} = (W_{out}^T + \Lambda(C^{-1}k_{\text{shortcut}})^T)^T, \tag{3.15}$$

where W_{out}^{new} is the new weights matrix that also contains the association between k_{shortcut} and $v_{\text{reprshift}}$, W_{out}^{T} is the transpose of W_{out}, $C = KK^{T}$, and

$$\Lambda = (v'_{\text{reprshift}} - W_{out}^{T} k_{\text{shortcut}}) / (C^{-1} k_{\text{shortcut}})^{T} k_{\text{shortcut}}$$

Finally, we set $W_{out} = W_{out}^{\text{new}}$ to modify the MLP in layer ℓ_{edit}.

REPRSHIFT Algorithm

Algorithm 7 REPRSHIFT

Function REPRSHIFT(ERM-trained model, validation dataset)
  Step 1: Compute Internal Representation for the Shortcut Attribute $k_{shortcut}$
  Step 2: Compute Logit Shift Vector $v_{logitshift}$
  Step 3: Compute Representation Shift Vector $v_{reprshift}$
  Step 4: Modify Model Weights such that, $k_{shortcut} \cdot W_{out}^{\text{new}} = v'_{reprshift}$
    and for all $k \neq k_{shortcut}$, $k \cdot W_{out}^{\text{new}} \approx k \cdot W_{out}^{\text{old}}$
  Return modified model

Chapter 4

Experiments

In this chapter, we present our experiments in two parts. After presenting our datasets for natural language inference (MultiNLI) and toxicity detection (CivilComments), in the first part, we inspect the internal representations of models fine-tuned with ERM and four different existing loss-function-based shortcut mitigation methods on MultiNLI and CivilComments via 3 experiments. Firstly, in Section 4.3.1, we compare the learned representations using Centered Kernel Alignment (CKA) (Kornblith et al., 2019). Then, in Section 4.3.2, we probe the representations for information about the shortcut attributes and labels. Thirdly, in Section 4.3.3, we investigate how the classifier layers use the information from the earlier layers to make predictions. In the second part of the chapter, we focus on models fine-tuned with ERM on MultiNLI. We conduct causal intervention experiments to understand how our model implements a shortcut rule (Section 4.4.1) and evaluate REPRSHIFT against the loss-function-based approaches (Section 4.4.2). The details of our fine-tuning setup is included in Appendix A.3. Examples from our datasets can be found in Appendix A.4.

4.1 Datasets

4.1.1 Natural Language Inference

The objective of an NLI task is to predict whether a premise contradicts, entails, or is neutral to a hypothesis. Table A.3 demonstrates examples from the MultiNLI (Idrissi et al., 2022; Williams et al., 2018) dataset, which is commonly used to study NLI. In MultiNLI, most of the examples where the hypothesis contains negation words “no”, “never”, “nobody” or “nothing” have contradiction label (Idrissi et al., 2022). Consequently, models trained on this dataset rely on negation words in the hypothesis as a shortcut to infer contradiction (Idrissi et al., 2022; Yang et al., 2023). We represent an example with attribute $a = 1$

when a negation word is present in the hypothesis and with attribute $a = 0$ otherwise. We denote contradiction as $y = 0$, entailment as $y = 1$, and neutral as $y = 2$. Under this setup, the MultiNLI dataset is divided into 6 data groups, one for each attribute-label pair (see Table A.3). The dataset is **label-balanced** and contains approximately equal number of examples from each of the three labels. MultiNLI has **attribute imbalance** as the examples that contain negation words are minority. Moreover, the dataset contains a **spurious correlation** since the examples that contain negation words are more likely to have contradiction labels. We use the same training ($206,175$ examples), validation ($82,462$ examples) and test ($123,712$ examples) splits from Idrissi et al. (2022) and Yang et al. (2023).

4.1.2 Toxicity Detection

Secondly, we study toxicity detection on the CivilComments (Borkan et al., 2019; Koh et al., 2021) dataset. The task is binary classification to predict whether an internet comment contains toxic language. We denote non-toxic label with $y = 0$ and toxic label with $y = 1$. The attributes are references to eight demographic identities: male ($a = 0$), female ($a = 1$), LGBTQ ($a = 2$), Christian ($a = 3$), Muslim ($a = 4$), other religions ($a = 5$), Black ($a = 6$), and White ($a = 7$). Under this setup, the CivilComments dataset is divided into 16 data groups for each attribute-label pair (see Table A.4). The dataset has **label imbalance** since non-toxic comments are more frequent than toxic comments. It also has **attribute imbalance** as the examples that contain references to certain demographic groups are more likely than others. For example, comments that have references to females are more frequent than those that reference males. Moreover, the dataset contains a **spurious correlations** since the examples that reference certain demographic groups are more likely to be toxic. For instance, examples that have LGBTQ references are more likely to be toxic compared to randomly sampled examples. We use the implementation of training ($148,304$ examples), validation ($24,278$ examples), and ($71,854$ examples) splits from the WILDS benchmark (Koh et al., 2021). Our implementation follows Yang et al. (2023) and differs from the binary attributes used by Idrissi et al. (2022).

4.2 Evaluation of Loss-Function-Based Approaches

Table 4.1 shows the accuracy (Acc.), worst-group accuracy (WGA, accuracy of the group with worst accuracy), and equally-weighted accuracy (EWA, average of group accuracies) for our BERT models fine-tuned on MultiNLI and CivilComments. We take WGA as our primary metric for comparison, as it provides a lower bound for the accuracy of the model

	Metric	ERM	GroupDRO	Focal	JTT	LFF
MNLI-Val	**Acc.**	81.6 (0.4)	81.1 (0.2)	81.2 (0.2)	80.8 (0.2)	51.3 (8.3)
	WGA	66.5 (1.3)	64.9 (1.3)	66.9 (1.6)	64.7 (2.4)	45.8 (21.5)
	EWA	71.6 (0.3)	70.5 (0.4)	70.4 (0.0)	69.2 (0.6)	47.0 (5.5)
MNLI-Test	**Acc.**	81.2 (0.6)	81.1 (0.2)	81.1 (0.3)	80.7 (0.1)	59.0 (2.7)
	WGA	64.6 (1.1)	64.3 (1.8)	65.6 (2.2)	64.3 (2.3)	46.1 (13.4)
	EWA	79.8 (0.1)	79.3 (0.2)	79.6 (0.4)	78.9 (0.0)	58.1 (2.9)
CC-Val	**Acc.**	84.2 (0.6)	81.2 (1.0)	84.5 (0.2)	83.2 (1.4)	39.7 (22.1)
	WGA	63.3 (2.1)	71.9 (1.5)	57.4 (1.4)	66.7 (6.2)	17.6 (3.5)
	EWA	72.5 (0.2)	74.0 (0.0)	71.7 (0.2)	74.2 (0.9)	46.1 (18.9)
CC-Test	**Acc.**	84.8 (1.2)	80.7 (0.9)	84.9 (0.5)	83.5 (1.7)	45.4 (21.4)
	WGA	66.2 (3.2)	72.8 (3.0)	63.4 (0.2)	69.6 (4.1)	14.5 (2.0)
	EWA	77.5 (0.3)	78.8 (0.5)	76.5 (0.1)	78.1 (0.4)	50.3 (19.1)

Table 4.1 Evaluation of Loss-Function-Based Approaches on MultiNLI (MNLI) and CivilComments (CC) validation (val) and test (test) sets. The reported values are averaged across the three runs and the standard deviations are reported in parentheses.

on OOD test sets where the proportion of the defined data groups can change arbitrarily. We observe that Focal Loss achieves slightly better worst-group accuracy compared to ERM on MultiNLI, while the other algorithms perform worse than ERM in terms of worst-group accuracy. On CivilComments, we observe that GroupDRO and JTT have better WGA compared to ERM-trained models. Notably, LFF consistently performs worse than both ERM and the other learning algorithms. We include detailed tables with group-specific accuracies in Appendix B. Next, to understand how the modifications in the loss function impact the training dynamics, we inspect the validation accuracies for each data group across training in Figures 4.1 and 4.2.

In Figure 4.1, we observe that the ERM-trained models achieve high accuracies in certain groups starting from the early stages of training. On the other hand, the accuracy for the worst group (the green line in Figure 4.1) seems to slowly improve as training progresses. We observe similar learning patterns for ERM, Focal Loss, and GroupDRO on the MultiNLI dataset, while the learning patterns of JTT and LFF differ from the other methods. In particular, we observe that LFF is effective in improving the accuracy of the group that has the worst performance in ERM-trained models (again, the green line in Figure 4.1). However, this comes at the cost of a substantial decrease in the accuracies of other groups, which explains the poor performance of the method.

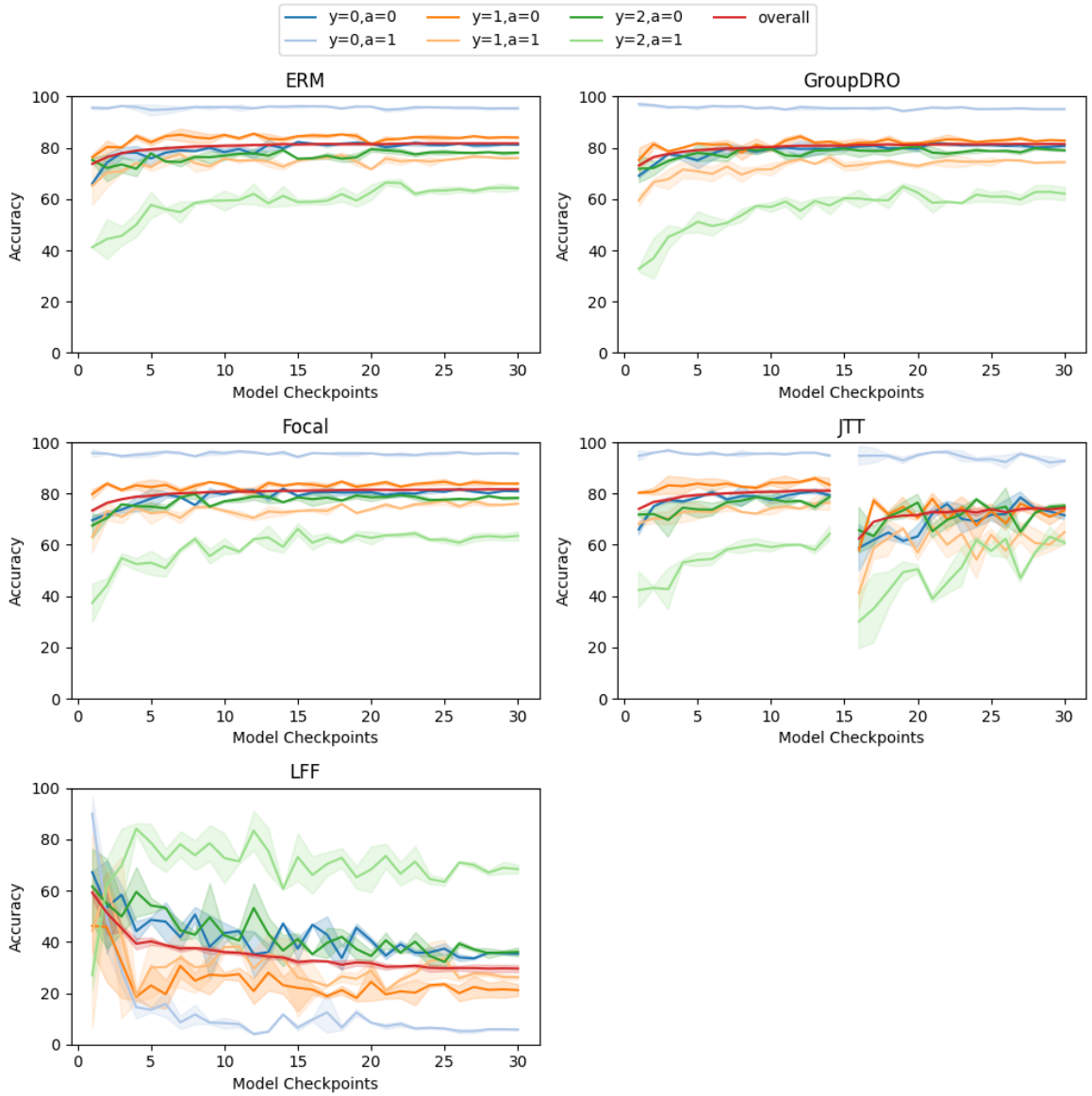


Fig. 4.1 MultiNLI Group Accuracy Across Training Checkpoints (Validation Set). The lines indicate the mean of the 3 training runs and the shading shows one standard deviation around the mean. JTT is missing checkpoint 15 because the method switches from stage-1 to stage-2 at checkpoint 15, halfway through the training.

In Figure 4.2, we clearly observe the implications of the label imbalance in the Civil-Comments dataset on group accuracies. Across training, the group accuracies can be seen in two separate bundles, which correspond to $y = 0$ (majority label has high accuracy) and $y = 1$ (minority label has low accuracy). We note that GroupDRO seems to be effective in addressing the label imbalance as it successfully reduces the discrepancy between the two bundles. This discrepancy can be clearly observed in models trained with ERM, Focal Loss,

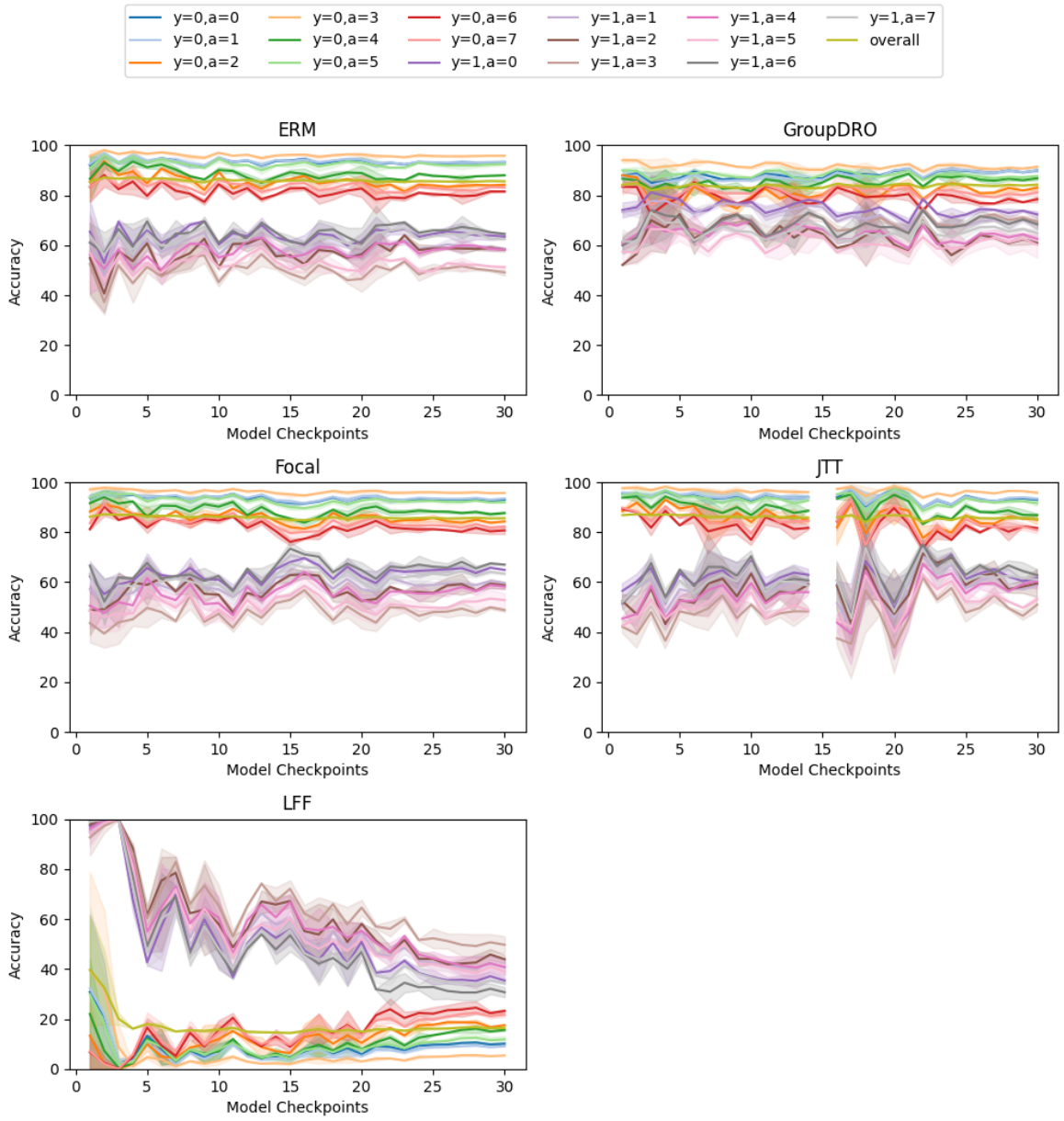


Fig. 4.2 CivilComments Group Accuracy Across Training Checkpoints (Validation Set). The lines indicate the mean of the 3 training runs and the shading shows one standard deviation around the mean. JTT is missing checkpoint 15 because the method switches from stage-1 to stage-2 at checkpoint 15, halfway through the training.

and JTT. Meanwhile, LFF is more effective in reducing the accuracy of the high-performing groups than improving the accuracy of the low-performing groups to close this gap.

Limitations

Gulrajani and Lopez-Paz (2020) and Idrissi et al. (2022) have observed that the performance of the loss-function-based approaches is sensitive to hyperparameter choices or model selection procedures. In our experiments, we follow the setup from Yang et al. (2023) for hyperparameter selection and use the *"best validation set worst-group accuracy"* as our model selection criteria. Similar to our findings, Yang et al. (2023) also did not observe worst-group-accuracy improvements over ERM in MultiNLI when they selected their final models with *"best validation set worst-group accuracy"* (Yang et al., 2023, Appendix E.2). The performance improvements were observed when they selected their models based on *"best* ***test set*** *worst group accuracy"* (Yang et al., 2023, Appendix E.1). We describe our fine-tuning setup in detail in Appendix A.3.

4.3 Understanding Representations

For the experiments in this section, we take a group-balanced random sample of 3600 examples from the test split of our datasets. Hence, the MultiNLI sample contains 600 examples from each group, and the CivilComments sample contains 225 examples from each group. Then, we run these examples through the model and save the internal representations that are generated by these examples. Precisely, we save the representations $r^{\ell}_{\text{CLS}} \in \mathbb{R}^{d_{model}}$ for $\ell \in \{1, 2, \ldots, 12\}$ (defined in Section 3.1.2). We refer to the representations from layer, say, 5 as r^{5}_{CLS}. Figure 3.2 demonstrates the role played by these representations in model computations. For 5 fine-tuning methods, for 12 layers in our model, and for the 3 training runs, we save $d_{model} = 768$ dimensional representations from 3600 examples. In total, we save $648,000$ ($5 \times 12 \times 3 \times 3600$) 768-dimensional representations for each dataset and $1,296,000$ ($648,000 \times 2$) representations in total.

4.3.1 Representation Similarity

Figures 4.3 and 4.4 show the results from our representation similarity experiments, in which we compare the internal representations of models trained with different loss functions. In addition to ERM, GroupDRO, Focal Loss, JTT, and LFF, we include 3 controls in this experiment. The first control is the representations generated by the pre-trained BERT model without any fine-tuning (pre-trained). Our second control is the representations from a randomly initialized model (randominit). This is a model whose weight terms are randomly chosen from a normal distribution with mean 0 and standard deviation 0.02. The bias terms

are set to 0. Thirdly, we include a set of random vectors that have the same size as the representations (random).

CKA: Intuition and Computation

We use Centered Kernel Alignment (CKA) (Kornblith et al., 2019) as our representation similarity metric. To compute CKA similarity between the representations of two models, Model A and Model B, at a specific layer ℓ, we stack the saved representations from 3600 examples in two matrices: $R_A \in \mathbb{R}^{3600 \times d_{model}}$ and $R_B \in \mathbb{R}^{3600 \times d_{model}}$, where the rows of R_A are the saved representations from Model A at layer ℓ. We then compute $K_A = R_A R_A^T$ and $K_B = R_B R_B^T$. Note that the ij^{th} element of K_A is the dot product of the rows i and j in R_A. Each row of R_A contains the representation for one of the 3600 examples. Therefore, ij^{th} element of K_A represents the similarity (dot product similarity in the case of CKA with Linear Kernel) between the representations of example i and example j from the perspective of Model A. More generally, the i^{th} row of K_A contains the representation similarities between example i and every other example in our sample of 3600 from the perspective of Model A. If we compare i^{th} row of K_A with the i^{th} row of K_B, we can understand how similar the models A and B in terms of how they perceive the similarity between example i and the other examples in our set of 3600. To obtain a single number that represents this similarity, we can take the dot product of i^{th} row of K_A with the i^{th} row of K_B. If we repeat this process for all examples in our dataset, we will obtain 3600 dot product similarities. To represent the similarity between the two models representations as a single number, we can take the sum of these 3600 numbers. We can represent this single number more compactly as the trace of the matrix $K_A K_B^T$. In addition to these steps, to compute CKA, we do three normalizations following (Kornblith et al., 2019). The first normalization is to account for the differences in the dimensions and scales of the representations of different models. After computing K_A, we normalize the columns of K_A by subtracting the column means from each column. We call this centered kernel $\tilde{K}_A$. Secondly, after we take the trace of $\tilde{K}_A \tilde{K}_B^T$, we divide the result by $(3600-1)^2$ to normalize for the number of examples in our batch (Kornblith et al., 2019). Notably, after this normalization, the trace becomes equivalent to the Frobenius norm of the cross-covariance matrix, $\text{cov}(R_A^T, R_B^T)$ by Hilbert-Schmidt Independence Criterion (HSIC) (Gretton et al., 2005; Kornblith et al., 2019). Therefore, this value is often called $\text{HSIC}(K_A, K_B)$. Thirdly, to obtain a number between 0 and 1 as the output of CKA similarity, we compute the final output as

$$\text{CKA}(R_A, R_B) = \frac{\text{HSIC}(R_A, R_B)}{\sqrt{\text{HSIC}(R_A, R_A) \cdot \text{HSIC}(R_B, R_B)}} \tag{4.1}$$

We experiment with both Linear and Radial Basis Function (RBF) Kernel CKA (Kornblith et al., 2019) and obtain similar results with both. Therefore, we report the results from the linear CKA we described above.

Observations

Figures 4.3 and 4.4 demonstrate pairwise CKA between the representations of our 5 models and 3 controls. Each square compares the representations from a particular layer, and the results are averaged across 3 training fine-tuning runs. We multiply CKA by 100 for better readability. In Figures 4.3 and 4.4, we make three observations:

- **Divergence from Pre-trained Representations:** In the early layers of the network, the representations generated by models fine-tuned with different methods are similar to the representations generated by a pre-trained model. In contrast, in the later layers of the network, the representations from fine-tuned models differ from those of the pre-trained model.
- **Similarity Across Layers in MultiNLI:** In MultiNLI, we observe substantial similarity in the later layers of the network between ERM, GroupDRO, Focal Loss, and JTT (Figure 4.3). This indicates that the loss-function-based approaches learn representations that are similar to ERM on MultiNLI.
- **Periodicity Across Layers in CivilComments:** In Figure 4.4, we observe that the representations from different fine-tuning methods differ in the mid-layers and become more similar again in the later layers, e.g., Layer 8 and Layer 11.

Overall, we find that the representations are more similar in the earlier layers, but patterns in representation similarity differ in the mid-to-late layers for different datasets. Most notably, in MultiNLI, we observe that different fine-tuning methods learn similar representations across all layers of the network, which provides an intuitive justification for the success of the last layer re-training (Kirichenko et al., 2023).

Related Work and Limitations

In our experiments, we used CKA to measure representation similarity. Other approaches, such as Canonical Correlation Analysis (CCA) (Bykhovskaya and Gorin, 2023; Guo and Wu, 2021; Zhou et al., 2023), which identifies linear combinations of features inside the representations that maximize the correlation between the representations from two models and measures the correlation between these linear projections, and Model Stitching (Bansal

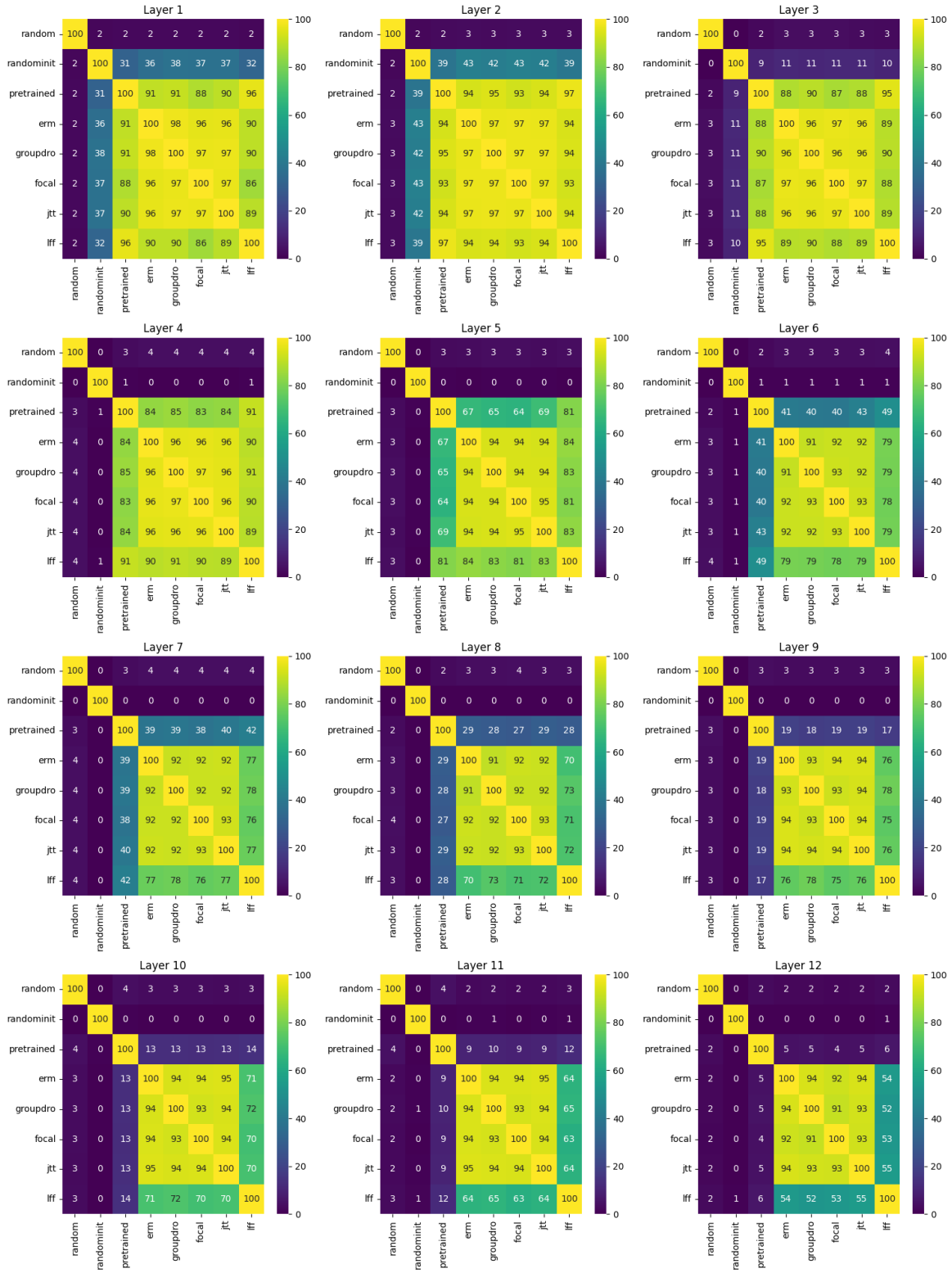


Fig. 4.3 MNLI Representation Similarity.

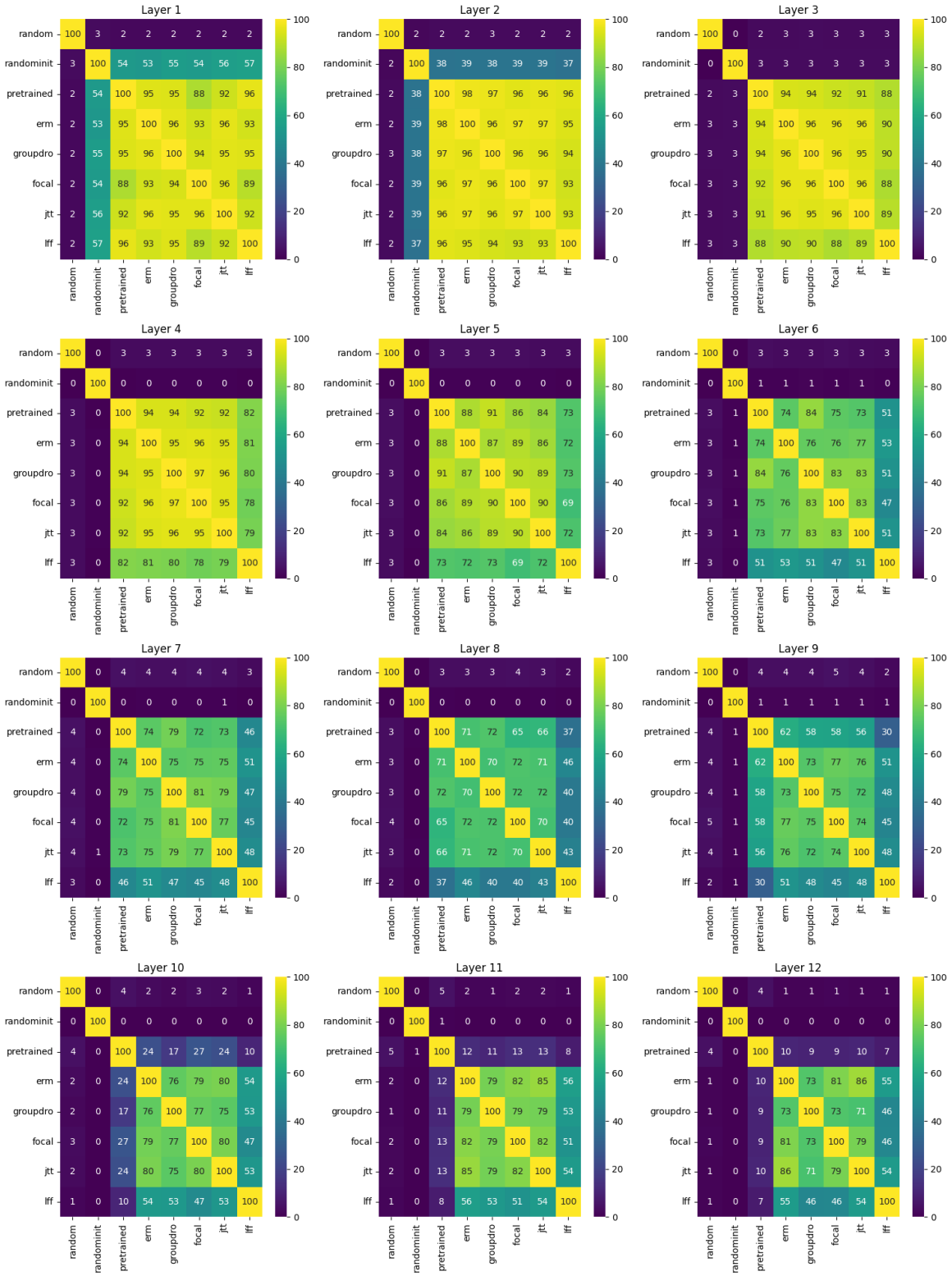


Fig. 4.4 CivilComments Representation Similarity.

et al., 2021; Hernandez et al., 2023), which replaces the layers from one model with the layers from another model and observes the change in the model's behavior after this change, have also been used to study the similarity between the representations of neural networks. Comparing the representations using different similarity metrics would be useful for a more comprehensive and robust analysis of the similarity between the representations learned by different fine-tuning approaches.

4.3.2 Information Contained in the Representations

Figures 4.6 and 4.5 show the results from our probing experiments. A probe is a classifier trained to predict some property from the representations (Belinkov and Glass, 2019; Rogers et al., 2020). In these experiments, our objective is to understand whether the representations $r^{\ell}_{\text{CLS}} \in \mathbb{R}^{d_{model}}$ contain information about (i) the attributes, a, (Shortcut Attribute Probe) or (ii) the labels, y, (Y Label Probe). Appendix A.5 describes the details of our experimental setup. We train both linear and two-layer MLP probes and find that they achieve similar probe accuracies. Therefore, here we report the results from our one-layer probes and include the results for the two-layer probes in the Appendix B. We use the same controls from our representation similarity experiments (Section 4.3.1).

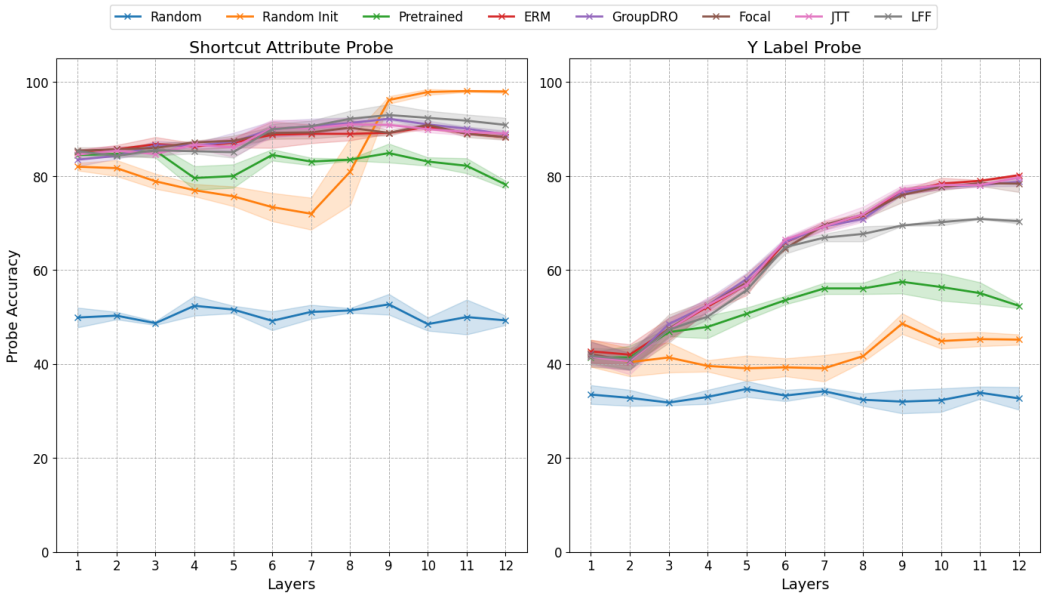


Fig. 4.5 MultiNLI Linear Probe. The lines indicate the mean of the 3 training runs and the shading shows one standard deviation around the mean.

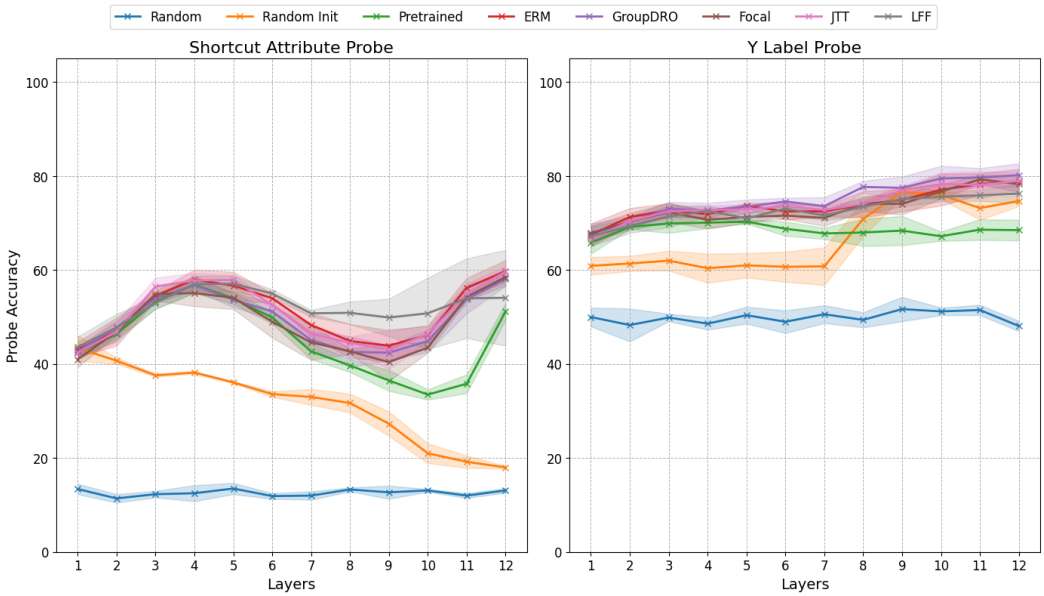


Fig. 4.6 CivilComments Linear Probe. The lines indicate the mean of the 3 training runs and the shading shows one standard deviation around the mean.

Observations

In Figures 4.5 and 4.6, we make the four observations:

- **ERM and Loss-Function-Based Approaches:** We find that the information contained in the representations is similar between ERM and other fine-tuning approaches (although LFF slightly differs).
- **Pre-trained and Fine-tuned Representations:** Probe accuracies for pre-trained representations differ from the fine-tuned models in the later layers of the network, which is consistent with our observations from 4.3.1. We observe that our fine-tuned models have higher Y label probe accuracies in the later layers of the network.
- **Binary Probe Target:** When the probe target is binary, probes on randomly initialized models also achieve high accuracies regardless of the task. For example, MultiNLI has binary attributes, and CivilComments has binary labels. In both of these cases, probes on randomly initialized models achieve similar accuracies to probes on fine-tuned models. However, probes on randomly initialized models perform worse compared to the fine-tuned and pre-trained models when the target is a multi-class classification, e.g., Y labels in MultiNLI and shortcut attributes in CivilComments.
- **Periodicity across Layers in CivilComments:** Interestingly, we observe a periodic pattern in CivilComments shortcut attribute probe accuracy for fine-tuned models. We

note that this resembles the periodic pattern in representation similarity observed in Section 4.3.1.

Overall, we find that the information contained in the representations from models fine-tuned with ERM is similar to the representations from models fine-tuned with other loss-function-based approaches.

Related Work and Limitations

Previous works have also studied the internal representations of transformers using probes (Hewitt and Liang, 2019; Rogers et al., 2020; Voita and Titov, 2020; Zhang and Bowman, 2018; Zhang et al., 2022). Similar to our findings, Zhang and Bowman (2018) have observed that probes on randomly initialized LSTM representations perform well on syntactic tagging tasks, and Hewitt and Liang (2019) have found that the accuracy of the probes can be similar when probing for linguistic labels and probing for randomly associated word types. Independent of the model weights, the representations contain information about the input data (hence attributes) because they are generated by passing input data through the model. Consequently, for simple classification tasks, this results in high probe accuracies. Voita and Titov (2020) has noted that reducing the amount of training data (Zhang and Bowman, 2018) or using smaller models (Hewitt and Liang, 2019) seems to help these issues and proposed minimum description length (MDL) probing, which incorporates the notions of (a) the final quality of the probe and (b) how hard it is to achieve that final quality. MDL probes, which compute the optimal bound on the *codelength* required to transmit the targets given the representations, can be used to further investigate the information contained in the representations. We leave this to future work.

4.3.3 Information Used by the Model

Next, to bridge the gap between "the information contained in the representations" and "the information used by the model to make predictions," we use a technique called Logit Lens (Belrose et al., 2023; Nostalgebraist, 2020).

Related Work

Logit Lens has been previously used to study the internal representations of GPT-style Transformers, where the logits of the model are a linear function of the representations in the last layer of the model. This linear function is often called the *unembedding matrix* (Belrose et al., 2023; Chughtai et al., 2023; Nanda et al., 2023a). Nostalgebraist (2020) has proposed

to apply the same linear function to the representations from the intermediate layers of the network to map the representations to the logit space. Previous studies have observed that this generates distributions that are interpretable (Belrose et al., 2023; Geva et al., 2022; Nostalgebraist, 2020; Ortu et al., 2024).

Logit Lens Setup

In BERT-style Transformers, different from GPT-style models, the logits are *not* a linear function of the representations at the last layer. Instead, the representations in the last layer, r^{12}_{CLS}, are mapped to the logits via two linear maps with a tanh activation function in between (see Equation 3.1.2). To obtain the logit distribution from the representations in the earlier layers of our network, we apply the classifier function from Equation 3.1.2 to the representations r^{ℓ}_{CLS} from intermediate layers. Then, we use these logits to get the predicted labels for the examples. Finally, we use these predicted labels to compute the accuracies for each data group. These group accuracies across layers are displayed in Figures 4.7 and 4.8.

Intuition

In section 3.1, we discussed the computations of the Transformer model around the idea of the residual stream (Elhage et al., 2021), where each layer 'read information from" and "write information to" the residual stream (Elhage et al., 2021). Due to the presence of residual connections around each layer, when we map the representations from the earlier layers to the logits using the classifier of the model, we obtain the predictions based on the information in the residual stream before the later layers get a chance to write additional information to the residual stream. This allows us to understand the contributions from which layers are influential for predicting each data group.

Observations

In Figure 4.7, we make the following observations:

- **Difference Across Labels:** Representations from early layers of the network often result in the predictions of "contradiction" and "neutral" labels. The features that make the model predict the entailment label seem to be added to the representations in the later layers of the network.
- **Difference Across Attributes:** Up to the 5^{th} layer of the network, we find that there is no difference between the accuracies of examples with and without shortcut attributes ($a = 0$ and $a = 1$) for each label. This suggests that the information the model uses

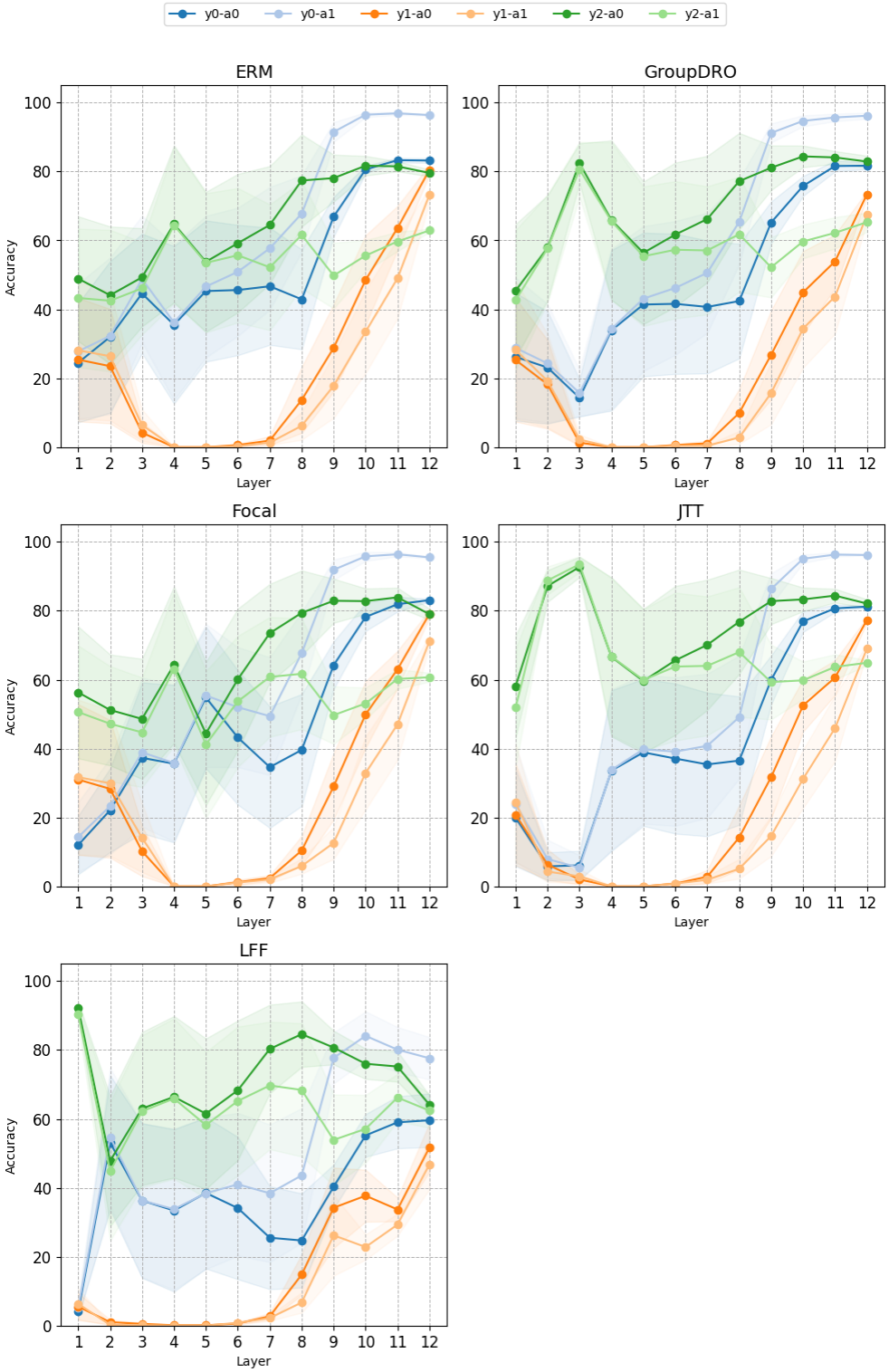


Fig. 4.7 MNLI Logit Lens. The lines indicate the mean of the 3 training runs and the shading shows 0.5 standard deviation around the mean.

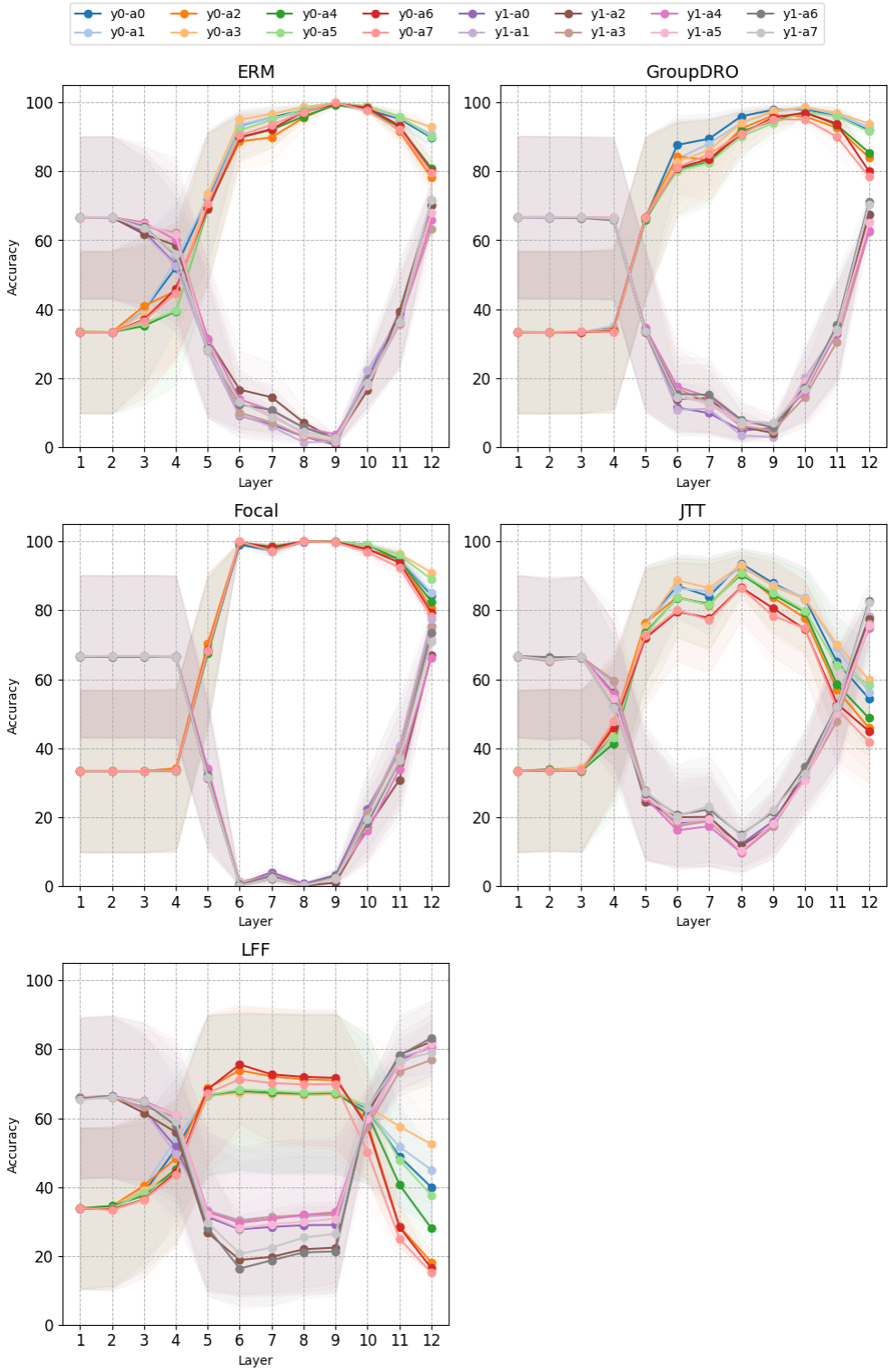


Fig. 4.8 CivilComments LogitLens. The lines indicate the mean of the 3 training runs and the shading shows 0.5 standard deviation around the mean.

to implement the shortcut rule (Algorithm 1) is added to the representations after the 5^{th} layer. After the 5^{th} layer, the accuracy for the examples with shortcut attributes ($a = 1$) diverges from the accuracy for the examples without shortcut attributes ($a = 0$): For examples with contradiction label ($y = 0$), accuracy of examples with shortcuts increases more steeply, whereas for the examples with neutral label ($y = 2$), accuracy of examples with shortcuts ($a = 1$) stays steady as the accuracy of examples without shortcuts ($a = 0$) increases slowly.

- **Difference Across Fine-Tuning Approaches:** There are differences in the patterns from different fine-tuning approaches. Since we have observed that their representations are similar, this suggests that the classifier layers differ.

In Figure 4.8, we observe:

- **Dominance of Label Imbalance:** Label imbalance in CivilComments can be clearly observed from the groups that are gathered in two bundles: non-toxic ($y = 0$) and toxic ($y = 1$). In the internal representations of the network, we observe that the influence of label imbalance dominates the impact of attribute imbalance and spurious correlations for this dataset.

- **Periodicity in per-Label Accuracies:** We find periodicity in per-label accuracies. In particular, in the early-mid (4 and 5) layers, the representation-classifier combination does not do better than random on the classification task, and there is no label imbalance. Then, in the mid-layers (4 to 9) label imbalance dominates the predictions. In the later layers (9 to 12), we see that the network balances the predictions as the accuracy for the majority label (non-toxic) examples decreases and the accuracy for the minority label examples increases.

- **Difference Across Fine-Tuning Approaches:** In ERM-trained models, as well as in our GroupDRO and Focal Loss models, the accuracies of the minority label come closer to the accuracy for examples in the majority label. In our models trained with JTT and LFF, we observe that the examples from the minority label are predicted with higher classification accuracy than those from the majority label when we use the representations from the later layers of the network. We also note that we obtain better worst group accuracies using the representations from the 11^{th} layer in our JTT model and 10^{th} layer in our LFF model (Figure 4.8).

In summary, we find that the predictions for the examples with and without shortcut attributes diverge in the later layers in MultiNLI, and label imbalance dominates the predictions in

CivilComments for mid-layers. Our findings also suggest that loss-function-based approaches change the classifier layers.

Limitations

Logit Lens assumes that the features are represented similarly in different layers. More precisely, it assumes that the directions in the d_{model} dimensional representation space have similar meanings in different layers of the model. To account for this *representational drift* (Belrose et al., 2023), Din et al. (2023) and Belrose et al. (2023) have recently proposed similar methodologies that apply different transformations to the representations from different layers to map the representations to the logits space. We leave exploring these approaches to future work.

4.4 Improving Representations

In section 4.3, we inspected the representations learned by ERM and different loss-function-based shortcut mitigation methods and found that, in MultiNLI, different loss-function-based approaches learn similar representations across all layers. In this section, we explore whether we can change the representations of a model fine-tuned with ERM on MultiNLI to improve the model's worst-group performance.

4.4.1 Causal Interventions

This section presents our causal intervention experiments (Pearl, 2013). In these experiments, we aim to understand the internal processes of models fine-tuned on MultiNLI with ERM at a greater depth before we modify the representations. We question CRH by investigating the internal channels through which the shortcut attributes impact the model's output logits.

Related Work

Causal interventions are used to identify model components that are responsible for specific model behaviors (Cammarata et al., 2020; Geva et al., 2022; Hanna et al., 2023; Li et al., 2023; Lieberum et al., 2023; Nanda et al., 2023b; Olsson et al., 2022; Varma et al., 2023; Wang et al., 2022; Wen et al., 2023; Zhong et al., 2023). For example, Vig et al. (2020) have studied the gender bias in GPT2 via causal interventions, and Meng et al. (2023a) used it to locate factual associations in GPT2. Lieberum et al. (2023) have used a similar method to analyze the internal processes of Chinchilla (Hoffmann et al., 2022). Other studies suggested

algorithms to automate causal intervention analysis to scale it for larger models (Conmy et al., 2023; Geiger et al., 2024; Wu et al., 2024).

Activation Patching Setup

Our goal in this experiment is to identify the channel inside the model through which the shortcut attribute impacts the logits. Firstly, we sample a set of examples that contain shortcut attributes from the validation split of MultiNLI. We use the validation split in this experiment because our insights inform our design decisions for REPRSHIFT, which we evaluate on the test split. In MultiNLI, the shortcut attributes can be represented as single tokens: "no", "never", "nobody", and "nothing". We call these *shortcut tokens*. We also define a corresponding *corrupted token* for each of the four shortcut tokens: "some" ("no"), "everything" ("nothing"), "everybody" ("nobody"), and "always" ("never").

Following the best practices outlined by Zhang and Nanda (2024), in our experiments, we implement causal interventions by patching the activations generated by a "clean run" with the activations from a "corrupted run". This process requires 3 forward passes through the model (Zhang and Nanda, 2024):

1. **Clean Run:** We pass the original input with the *shortcut token* through the model.
2. **Corrupted Run:** We replace the *shortcut token* in the original input with the corresponding *corrupted token* and pass this *corrupted input* through the model.
3. **Patched Run:** We repeat the corrupted run but replace the activations outputted by a specific model component with the activations outputted by the same model component in the clean run.

To measure how much the predictions of our model change between the clean run, the corrupted run, and the patched run, we use average logit difference (Zhang and Nanda, 2024) as our metric. More precisely, each of our three runs generate a 3 dimensional logit output: $l^{\text{cl}}, l^{\text{co}}, l^{\text{pa}} \in \mathbb{R}^3$. In the validation set, we observe that the presence of the shortcut tokens tends to increase the contradiction logits and decrease the logits for the other two labels (Table 3.2). Therefore, we define logit difference as

$$\text{LogitDiff}(l) = l_0 - (l_1 + l_2), \tag{4.2}$$

where l_0 is the logit that corresponds to the contradiction label; l_1 and l_2 are logits for entailment and neutral labels, respectively. By taking the difference between the contradiction logit and the sum of the other two logits, this metric captures how confident the model is in predicting the contradiction label.

	Clean Run	Corrupted Run	Control Run
Average Logit Difference	5.38	-2.29	3.79

Table 4.2 Average Logit Difference. Reported Results are the Averages across 300 examples randomly sampled from the MultiNLI validation split. The results are averaged across 3 training runs.

Table 4.2 shows the average of the logit differences we calculate for the clean and the corrupted runs. We observe that the average logit difference is high in the clean runs, and it significantly decreases in the corrupted runs. We also add a control run, where the input is the same as the clean run except for a randomly selected token that is replaced by one of the 4 corrupted tokens. This control aims to check that the decrease in the logit difference from clean to corrupted run is mostly due to the removal of the shortcut token from the input rather than the injection of the corrupt token into the input. Finally, to measure how much of the difference between the logit differences in the clean and corrupted runs is recovered by the patched run, we compute

$$\text{LogitDiffRecovered}(l^{\text{cl}}, l^{\text{co}}, l^{\text{pa}}) = \frac{\text{LogitDiff}(l^{\text{pa}}) - \text{LogitDiff}(l^{\text{co}})}{\text{LogitDiff}(l^{\text{cl}}) - \text{LogitDiff}(l^{\text{co}})} \tag{4.3}$$

This metric allows us to quantify the impact of patching a specific set of activations. In particular, it allows us to measure how much the patched activations increase the model's contradiction logits and decrease the other logits.

Procedure

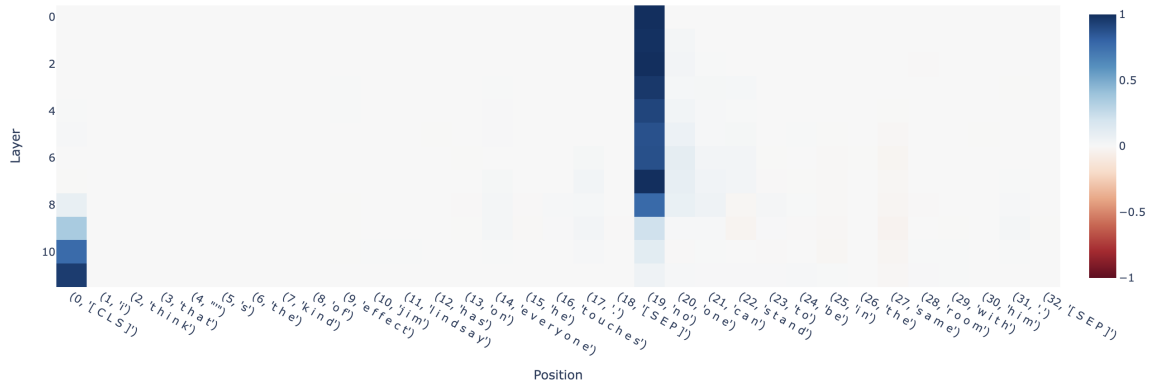


Fig. 4.9 Activation Patching Example 1. (The Figure displays only the first 33 positions in the context. The rest of the context consists of padding tokens.)

Figure 4.9 shows the results of activation patching for a single example, where the premise is "i think that's the kind of effect jim lindsay has on everyone he touches." and the hypothesis is " no one can stand to be in the same room with him.". We change the hypothesis to be " some one can stand to be in the same room with him." in the corrupted run. The true label of the example is neutral. The clean run results in the logits $l_0^{\text{cl}} = 1.83, l_1^{\text{cl}} = -3.56, l_2^{\text{cl}} = 1.47$. Consequently, the example is misclassified as contradiction. In the corrupted run, we obtain $l_0^{\text{cl}} = -1.86, l_1^{\text{cl}} = 0.44, l_2^{\text{cl}} = 1.71$, which results in the correct classification of the example as neutral. The cell colors in the figure indicate the value of Logit Difference Recovered (Equation 4.4.1) after patching the activation at a specific layer and specific position. The first column in Figure 4.9 demonstrates the impact of patching the representation r_{CLS}^{ℓ} for $\ell \in \{1, 2, \ldots, 12\}$. In the first column, at each row, the patching for r_{CLS}^{ℓ} is done for a different layer ℓ. The remaining columns represent the impact of patching different rows from matrix $r_{\ell} \in \mathbb{R}^{d_{context} \times d_{model}}$. We refer the reader to Section 3.1.2 for the precise definition of r_{ℓ}. To obtain the Logit Difference Recovered values for each position in context and for each of the 12 layers, we do $d_{context} \times 12 = 128 \times 12 = 1536$ patching runs. Figure 4.10 demonstrates the same process for an example whose true label is contradiction.

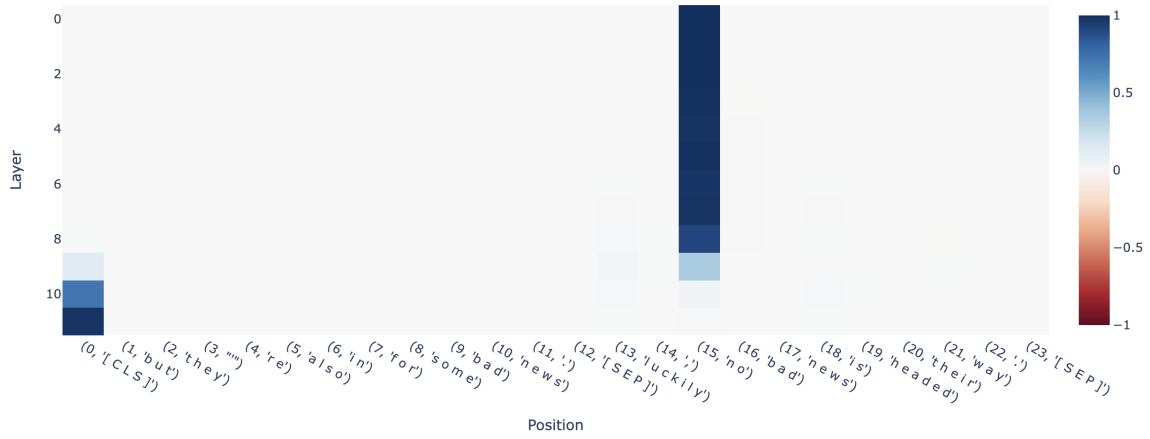


Fig. 4.10 Activation Patching Example 2. (The Figure displays only the first 24 positions in the context. The rest of the context consists of padding tokens.)

Figure 4.11 demonstrates the aggregated results from patching 100 examples for two separate models from different fine-tuning runs with ERM. The results from a third model is also included in the Appendix. The first column represents the effect of patching r_{CLS}^{ℓ}. It corresponds to the first column in Figures 4.9 and 4.10. The second column represents the sum of the remaining 127 positions in the context.

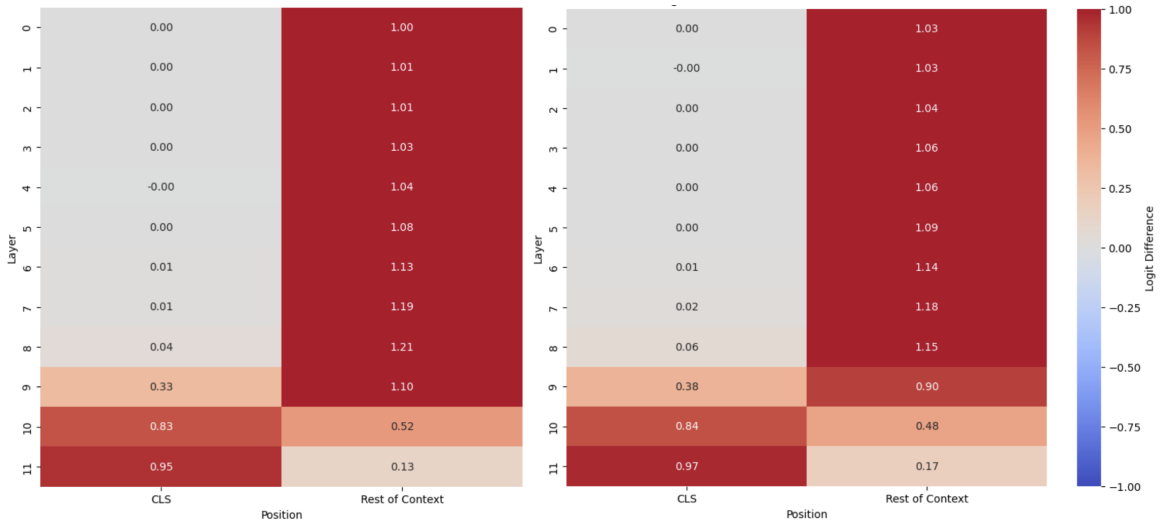


Fig. 4.11 Patching r_ℓ. Two models from separate training runs. For each model, we average the results across 100 examples.

Observations

In Figures 4.9, 4.10, and 4.11, we make two important observations.

- **Shortcut Signal Channel:** In Figure 4.11), we observe that the information about the shortcut attribute travels through the 1st to the 9th layers of the model in positions other than the [CLS] token position. Then, it is moved to the position of the [CLS] token around the 10th layer. After this information is moved from the position of the shortcut token to the position of the [CLS] token, patching the representations at the position of the shortcut token does not impact the model's predictions (We observe this via close to 0 Logit Difference recovered values). Similarly, patching r^ℓ_{CLS} for $\ell < 10$ does not impact the models predictions. Based on this observation, in our implementation of REPRSHIFT, we make modifications on r^ℓ_{CLS} for $\ell < 10$. Our reported results focus on edits on the Transformer layer that takes r^{11} as input and outputs r^{12} (Section 4.4.2).

- **Narrow Channels (Suggestive Evidence for CRH):** Secondly, in Figures 4.9 and 4.10, we observe that the shortcut signal travels though the model through a narrow channel: only via the shortcut token position in the early layers, then the [CLS] token position in the later layers. Our manual inspection of different examples reveals that this is a common pattern. In these examples, the impact of the shortcut token on the logits does not seem to be informed by interactions between shortcut token an the other tokens in the premise and the hypothesis since patching any of the other positions in the context does not impact the logits of the model. These ***narrow channels*** suggest that

the shortcut token is directly impacting the logits of the model, as described by CRH (Section 3.2). We also note that in examples where the hypothesis contains another negation word, such as "worse", that we have not included in our list of shortcut tokens, the patching patterns deviate from the narrow channels. An example for this phenomenon can be seen in Figure 4.12.

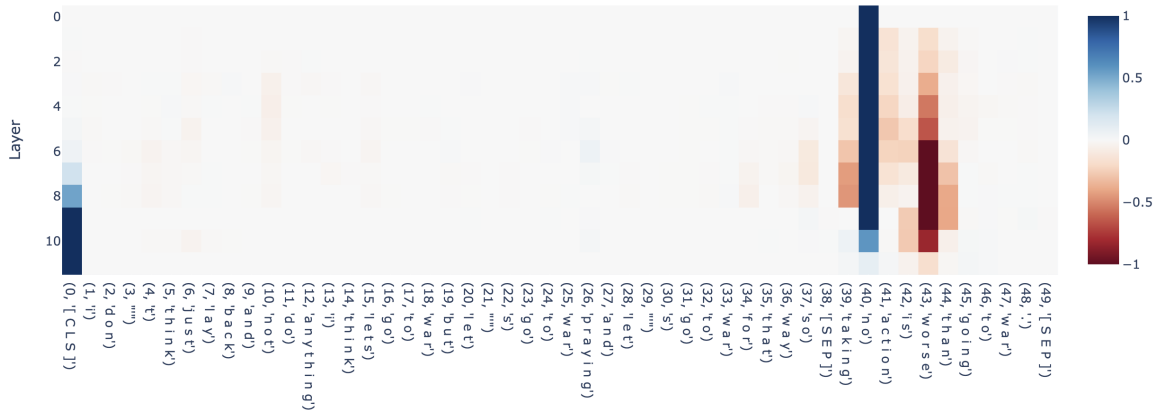


Fig. 4.12 A counterexample to *narrow channels*.

4.4.2 Evaluating REPRSHIFT

	ERM	GroupDRO	Focal	JTT	LFF	REPRSHIFT
Overall	81.2 (0.6)	81.1 (0.2)	81.1 (0.3)	80.7 (0.1)	59.0 (2.7)	79.8 (0.2)
WGA	64.6 (1.1)	64.3 (1.8)	65.6 (2.2)	64.3 (2.3)	46.1 (13.4)	**72.9 (3.1)**
EWA	79.8 (0.1)	79.3 (0.2)	79.6 (0.4)	78.9 (0.0)	58.1 (2.9)	79.5 (0.1)
y=0, a=0	79.2 (1.2)	80.1 (0.5)	79.2 (0.6)	78.1 (0.4)	58.3 (16.5)	77.8 (0.3)
y=0, a=1	94.5 (0.8)	94.2 (0.2)	94.7 (0.5)	94.8 (0.4)	76.6 (13.1)	88.9 (1.7)
y=1, a=0	83.2 (0.9)	81.1 (0.9)	82.0 (1.4)	81.9 (0.8)	52.1 (16.6)	83.0 (0.7)
y=1, a=1	77.0 (1.3)	73.6 (1.7)	75.0 (0.3)	74.8 (1.0)	46.1 (13.4)	77.3 (1.4)
y=2, a=0	79.3 (0.5)	80.3 (1.4)	80.2 (0.5)	80.1 (0.4)	64.0 (6.2)	77.0 (0.2)
y=2, a=1	64.6 (1.1)	64.3 (1.8)	65.6 (2.2)	64.3 (2.3)	62.0 (10.7)	72.9 (3.1)

Table 4.3 Evaluation on MultiNLI Test Split. The reported results are the average of 3 training runs and the standard deviations are reported in parenthesis.

Table 4.3 demonstrates the results from our evaluation of different loss-function-based approaches against REPRSHIFT on MultiNLI. When we modify the last Transformer layer

of an ERM-trained model with REPRSHIFT, the model achieves substantially better worst-group accuracies compared to ERM and the other loss-function-based approaches. This hints at promising prospects for addressing shortcut learning with interpretability-based approaches. However, two critical caveats make a direct comparison between REPRSHIFT and loss-function-based approaches unfair. Firstly, loss-function-based approaches do not make assumptions about the functional form of the neural network, i.e., they are model agnostic and can be used with models other than Transformers. Secondly, Focal Loss, JTT, and LFF do not require the shortcut attributes to be known during training. Therefore, they can, in theory, address shortcut learning in more general settings. Among the loss-function-based approaches we studied, only GroupDRO requires knowledge of shortcut attributes during training. Table 4.3 should be treated as a proof-of-concept for REPRSHIFT, rather than an equal comparison between different approaches.

Why REPRSHIFT Works?

To understand the reason behind the effectiveness of REPRSHIFT, we manually inspect the logits of our model before and after the model is modified. We identify a common pattern and explain the intuition behind it in Figures 4.13 and 4.14. For these examples, assume that we have chosen $v_{\text{logitshift}} = [-4, 0, 0]$ following the procedure described in Section 3.3.

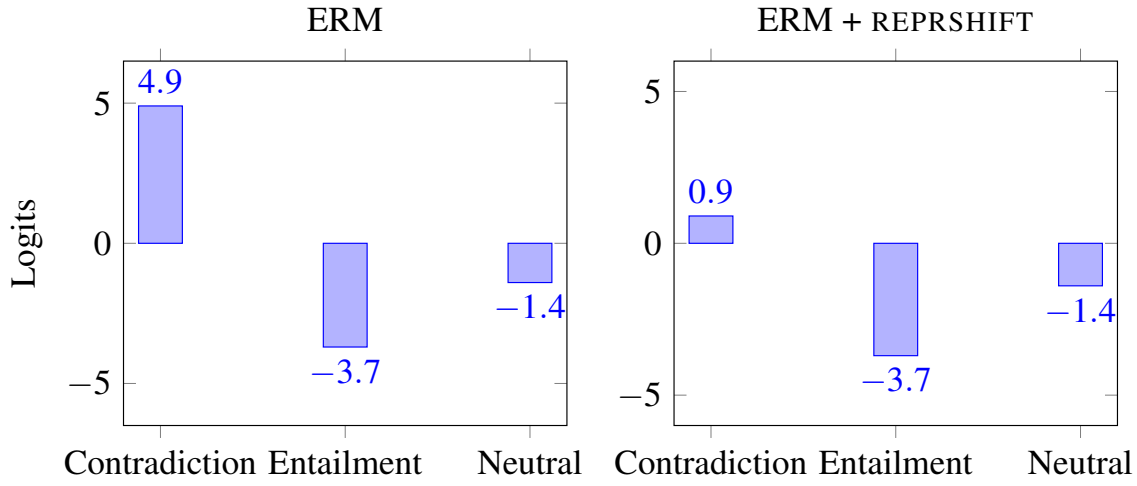


Fig. 4.13 Impact of REPRSHIFT. **True Label:** Contradiction. **Premise:** "but they're also in for some bad news." **Hypothesis:** "luckily, no bad news is headed their way."

In Figure 4.13, we demonstrate the impact of the logit shift implemented by REPRSHIFT on the logits of an example with contradiction label and shortcut attribute. Examples with this attribute-label combination have especially high contradiction logits due to constructive interference (defined in Section 3.2.2). Therefore, adding $v_{\text{logitshift}}$ to the logits does not

change the predicted label of the example: The example is classified correctly before and after REPRSHIFT.

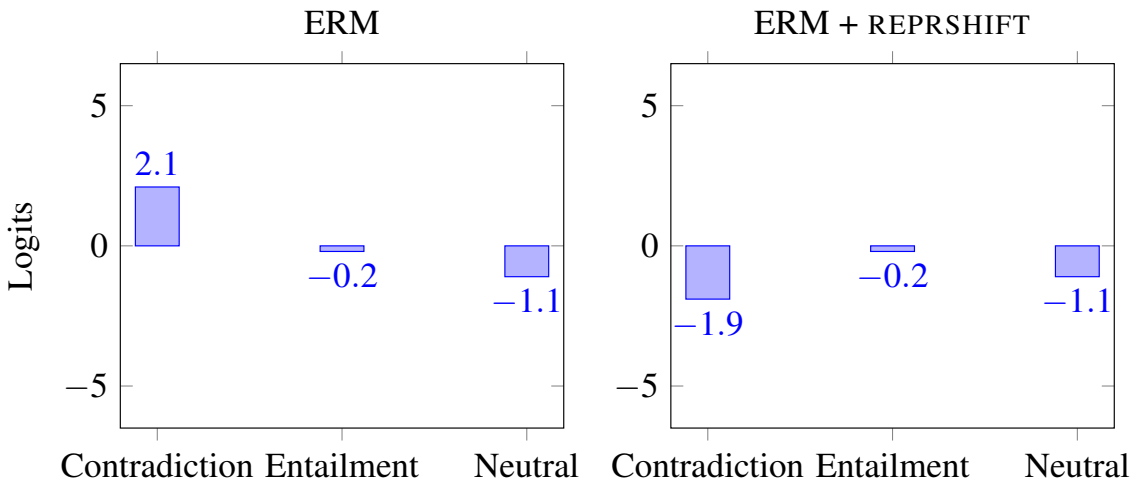


Fig. 4.14 Impact of REPRSHIFT . **Label:** Entailment. **Premise:** "for all there is to do and see in las vegas, one may wonder if there is a nightlife at all outside of the obvious." **Hypothesis:** "there's no shortage of things to do in las vegas."

In Figure 4.14, we consider an example with a shortcut attribute and entailment label. The example is misclassified as a contradiction by ERM. When we add $v_{\text{logitshift}}$ to the logits of the example, the contradiction logit is suppressed, and the example is correctly classified to be entailment. We also note that, for examples that do not contain shortcut attributes, adding $v_{\text{logitshift}}$ to the logits leads to worse performance (This phenomenon can be observed in Figure A.1).

Limitations and Future Work

A major limitation of our work is the limited scope of our evaluation of REPRSHIFT. To establish the effectiveness of our method, evaluation across different datasets is necessary. Moreover, we only tested the performance of REPRSHIFT for $\ell = 12$, limiting our consideration to the modifications to the last Transformer layer of the network. In future work, we aim to conduct more comprehensive evaluations of REPRSHIFT across different datasets and layers. Investigating the impact of REPRSHIFT on the representations is another promising avenue.

Chapter 5

Conclusion

In this work, we studied representation learning in the presence of shortcuts. We inspected the representations learned by ERM and different loss-function-based shortcut mitigation methods and found that, in MultiNLI, different loss-function-based approaches learn similar representations across all layers (Section 4.3). We introduced CRH as a framework to understand shortcut learning and provided three lines of suggestive evidence in support of this hypothesis (Sections 3.2.2 and 4.4.1). Building on CRH, we developed REPRSHIFT as an interpretability-based method that shifts the representations of the network to mitigate shortcuts (Section 3.3). In our evaluations, we observed that the loss-function-based approaches often do not lead to worst-group accuracy improvements under our setup (Section 4.2). In comparison, by shifting the representations, we achieved substantial improvements in worst-group accuracy (Section 4.4.2).

Future work can conduct more comprehensive evaluations of REPRSHIFT and build on our approach to explore its applications in more general settings where the shortcut attributes are not known during training. We believe that this can be done in two steps by, firstly, training a "shortcut model" to capture the spurious correlations in the dataset and, secondly, applying REPRSHIFT to an ERM-trained model to remove the rule learned by the "shortcut model". Additionally, investigating scenarios where the training data contains multiple shortcuts is a promising direction. Furthermore, REPRSHIFT can be a complement to different fine-tuning approaches rather than a competitor, working as a second step after fine-tuning to remove shortcuts.

We hope that our work demonstrates the potential benefits of opening the black box to understand the model computations and representations. In particular, we highlight that interpretability is not only beneficial for understanding the model but can also be used to modify the model weights to remove an undesired model behavior, in our case, shortcut learning.

References

Anderson, J. A. (1972). A simple neural network generating an interactive memory. *Mathematical Biosciences*, 14(3):197–220.

Arjovsky, M., Bottou, L., Gulrajani, I., and Lopez-Paz, D. (2020). Invariant risk minimization.

Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization.

Bansal, Y., Nakkiran, P., and Barak, B. (2021). Revisiting model stitching to compare neural representations.

Bau, D., Liu, S., Wang, T., Zhu, J.-Y., and Torralba, A. (2020). Rewriting a deep generative model.

Belinkov, Y. and Glass, J. (2019). Analysis methods in neural language processing: A survey. *Transactions of the Association for Computational Linguistics*, 7:49–72.

Belrose, N., Furman, Z., Smith, L., Halawi, D., Ostrovsky, I., McKinney, L., Biderman, S., and Steinhardt, J. (2023). Eliciting latent predictions from transformers with the tuned lens.

Blodgett, S. L., Green, L., and O'Connor, B. (2016). Demographic dialectal variation in social media: A case study of African-American English. In Su, J., Duh, K., and Carreras, X., editors, *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1119–1130, Austin, Texas. Association for Computational Linguistics.

Borkan, D., Dixon, L., Sorensen, J., Thain, N., and Vasserman, L. (2019). Nuanced metrics for measuring unintended bias with real data for text classification.

Bykhovskaya, A. and Gorin, V. (2023). High-dimensional canonical correlation analysis.

Cai, T., Gao, R., Lee, J., and Lei, Q. (2021). A theory of label propagation for subpopulation shift. In Meila, M. and Zhang, T., editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 1170–1182. PMLR.

Cammarata, N., Carter, S., Goh, G., Olah, C., Petrov, M., Schubert, L., Voss, C., Egan, B., and Lim, S. K. (2020). Thread: Circuits. *Distill*. https://distill.pub/2020/circuits.

Cao, K., Wei, C., Gaidon, A., Arechiga, N., and Ma, T. (2019). Learning imbalanced datasets with label-distribution-aware margin loss.

Chughtai, B., Chan, L., and Nanda, N. (2023). A toy model of universality: Reverse engineering how networks learn group operations.

Conmy, A., Mavor-Parker, A. N., Lynch, A., Heimersheim, S., and Garriga-Alonso, A. (2023). Towards automated circuit discovery for mechanistic interpretability.

Cortiz, D. (2021). Exploring transformers in emotion recognition: a comparison of bert, distillbert, roberta, xlnet and electra.

Creager, E., Jacobsen, J.-H., and Zemel, R. (2021). Environment inference for invariant learning.

Cui, Y., Jia, M., Lin, T.-Y., Song, Y., and Belongie, S. (2019). Class-balanced loss based on effective number of samples.

De Cao, N., Aziz, W., and Titov, I. (2021). Editing factual knowledge in language models. In Moens, M.-F., Huang, X., Specia, L., and Yih, S. W.-t., editors, *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 6491–6506, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

DeGrave, A., Janizek, J., and Lee, S.-I. (2021). Ai for radiographic covid-19 detection selects shortcuts over signal. *Nature Machine Intelligence*, 3.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In Burstein, J., Doran, C., and Solorio, T., editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Din, A. Y., Karidi, T., Choshen, L., and Geva, M. (2023). Jump to conclusions: Short-cutting transformers with linear transformations.

Du, M., He, F., Zou, N., Tao, D., and Hu, X. (2023). Shortcut learning of large language models in natural language understanding.

Duchi, J., Glynn, P., and Namkoong, H. (2018). Statistics of robust optimization: A generalized empirical likelihood approach.

Duchi, J., Hashimoto, T., and Namkoong, H. (2022). Distributionally robust losses for latent covariate mixtures.

Duchi, J. and Namkoong, H. (2020). Learning models with uniform performance via distributionally robust optimization.

Elhage, N., Nanda, N., Olsson, C., Henighan, T., Joseph, N., Mann, B., Askell, A., Bai, Y., Chen, A., Conerly, T., DasSarma, N., Drain, D., Ganguli, D., Hatfield-Dodds, Z., Hernandez, D., Jones, A., Kernion, J., Lovitt, L., Ndousse, K., Amodei, D., Brown, T., Clark, J., Kaplan, J., McCandlish, S., and Olah, C. (2021). A mathematical framework for transformer circuits. *Transformer Circuits Thread*. https://transformer-circuits.pub/2021/framework/index.html.

Geiger, A., Wu, Z., Potts, C., Icard, T., and Goodman, N. D. (2024). Finding alignments between interpretable causal variables and distributed neural representations.

Geirhos, R., Jacobsen, J.-H., Michaelis, C., Zemel, R., Brendel, W., Bethge, M., and Wichmann, F. A. (2020). Shortcut learning in deep neural networks. *Nature Machine Intelligence*, 2(11):665–673.

Geva, M., Caciularu, A., Wang, K., and Goldberg, Y. (2022). Transformer feed-forward layers build predictions by promoting concepts in the vocabulary space. In Goldberg, Y., Kozareva, Z., and Zhang, Y., editors, *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 30–45, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Gowda, S. C. M., Joshi, S., Zhang, H., and Ghassemi, M. (2021). Pulling up by the causal bootstraps: Causal data augmentation for pre-training debiasing. In *Proceedings of the 30th ACM International Conference on Information amp; Knowledge Management*, CIKM '21. ACM.

Gretton, A., Bousquet, O., Smola, A., and Scholkopf, B. (2005). Measuring statistical dependence with hilbert-schmidt norms. In *International Conference on Algorithmic Learning Theory*.

Gulrajani, I. and Lopez-Paz, D. (2020). In search of lost domain generalization.

Guo, C. and Wu, D. (2021). Canonical correlation analysis (cca) based multi-view learning: An overview.

Han, Z., Liang, Z., Yang, F., Liu, L., Li, L., Bian, Y., Zhao, P., Wu, B., Zhang, C., and Yao, J. (2023). Umix: Improving importance weighting for subpopulation shift via uncertainty-aware mixup.

Hanna, M., Liu, O., and Variengien, A. (2023). How does gpt-2 compute greater-than?: Interpreting mathematical abilities in a pre-trained language model.

Hase, P., Diab, M., Celikyilmaz, A., Li, X., Kozareva, Z., Stoyanov, V., Bansal, M., and Iyer, S. (2021). Do language models have beliefs? methods for detecting, updating, and visualizing model beliefs.

Hashimoto, T. B., Srivastava, M., Namkoong, H., and Liang, P. (2018). Fairness without demographics in repeated loss minimization.

Hendrycks, D. and Gimpel, K. (2023). Gaussian error linear units (gelus).

Hernandez, A., Dangovski, R., Lu, P. Y., and Soljacic, M. (2023). Model stitching: Looking for functional similarity between representations.

Hewitt, J. and Liang, P. (2019). Designing and interpreting probes with control tasks. In Inui, K., Jiang, J., Ng, V., and Wan, X., editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2733–2743, Hong Kong, China. Association for Computational Linguistics.

Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., de Las Casas, D., Hendricks, L. A., Welbl, J., Clark, A., Hennigan, T., Noland, E., Millican, K., van den Driessche, G., Damoc, B., Guy, A., Osindero, S., Simonyan, K., Elsen, E., Rae, J. W., Vinyals, O., and Sifre, L. (2022). Training compute-optimal large language models.

Hovy, D. and Søgaard, A. (2015). Tagging performance correlates with author age. In Zong, C. and Strube, M., editors, *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 483–488, Beijing, China. Association for Computational Linguistics.

Idrissi, B. Y., Arjovsky, M., Pezeshki, M., and Lopez-Paz, D. (2022). Simple data balancing achieves competitive worst-group-accuracy.

Izmailov, P., Kirichenko, P., Gruver, N., and Wilson, A. G. (2022). On feature learning in the presence of spurious correlations.

Joshi, N., Pan, X., and He, H. (2022). Are all spurious features in natural language alike? an analysis through a causal lens. In Goldberg, Y., Kozareva, Z., and Zhang, Y., editors, *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 9804–9817, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Kalyan, K. S., Rajasekharan, A., and Sangeetha, S. (2021). Ammus : A survey of transformer-based pretrained models in natural language processing.

Kang, B., Xie, S., Rohrbach, M., Yan, Z., Gordo, A., Feng, J., and Kalantidis, Y. (2020). Decoupling representation and classifier for long-tailed recognition.

Kirichenko, P., Izmailov, P., and Wilson, A. G. (2023). Last layer re-training is sufficient for robustness to spurious correlations.

Koh, P. W., Sagawa, S., Marklund, H., Xie, S. M., Zhang, M., Balsubramani, A., Hu, W., Yasunaga, M., Phillips, R. L., Gao, I., Lee, T., David, E., Stavness, I., Guo, W., Earnshaw, B., Haque, I., Beery, S. M., Leskovec, J., Kundaje, A., Pierson, E., Levine, S., Finn, C., and Liang, P. (2021). Wilds: A benchmark of in-the-wild distribution shifts. In Meila, M. and Zhang, T., editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 5637–5664. PMLR.

Kohonen, T. (1972). Correlation matrix memories. *IEEE Transactions on Computers*, C-21:353–359.

Kornblith, S., Norouzi, M., Lee, H., and Hinton, G. (2019). Similarity of neural network representations revisited.

Li, K., Hopkins, A. K., Bau, D., Viégas, F., Pfister, H., and Wattenberg, M. (2023). Emergent world representations: Exploring a sequence model trained on a synthetic task.

Li, Q., Peng, H., Li, J., Xia, C., Yang, R., Sun, L., Yu, P. S., and He, L. (2021). A survey on text classification: From shallow to deep learning.

Li, T., Cao, P., Yuan, Y., Fan, L., Yang, Y., Feris, R., Indyk, P., and Katabi, D. (2022). Targeted supervised contrastive learning for long-tailed recognition.

Lieberum, T., Rahtz, M., Kramár, J., Nanda, N., Irving, G., Shah, R., and Mikulik, V. (2023). Does circuit analysis interpretability scale? evidence from multiple choice capabilities in chinchilla.

Lin, T.-Y., Goyal, P., Girshick, R., He, K., and Dollár, P. (2018). Focal loss for dense object detection.

Liu, E. Z., Haghgoo, B., Chen, A. S., Raghunathan, A., Koh, P. W., Sagawa, S., Liang, P., and Finn, C. (2021). Just train twice: Improving group robustness without training group information.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2019). Roberta: A robustly optimized bert pretraining approach.

Meng, K., Bau, D., Andonian, A., and Belinkov, Y. (2023a). Locating and editing factual associations in gpt.

Meng, K., Sen Sharma, A., Andonian, A., Belinkov, Y., and Bau, D. (2023b). Mass editing memory in a transformer. *The Eleventh International Conference on Learning Representations (ICLR)*.

Min, B., Ross, H., Sulem, E., Veyseh, A. P. B., Nguyen, T. H., Sainz, O., Agirre, E., Heinz, I., and Roth, D. (2021). Recent advances in natural language processing via large pre-trained language models: A survey.

Mitchell, E., Lin, C., Bosselut, A., Finn, C., and Manning, C. D. (2022a). Fast model editing at scale.

Mitchell, E., Lin, C., Bosselut, A., Manning, C. D., and Finn, C. (2022b). Memory-based model editing at scale.

Nam, J., Cha, H., Ahn, S., Lee, J., and Shin, J. (2020). Learning from failure: Training debiased classifier from biased classifier.

Nam, J., Kim, J., Lee, J., and Shin, J. (2022). Spread spurious attribute: Improving worst-group accuracy with spurious attribute estimation.

Nanda, N. and Bloom, J. (2022). Transformerlens. https://github.com/TransformerLensOrg/TransformerLens.

Nanda, N., Chan, L., Lieberum, T., Smith, J., and Steinhardt, J. (2023a). Progress measures for grokking via mechanistic interpretability.

Nanda, N., Lee, A., and Wattenberg, M. (2023b). Emergent linear representations in world models of self-supervised sequence models.

Nostalgebraist (2020). interpreting gpt: the logit lens. *LessWrong*.

Olsson, C., Elhage, N., Nanda, N., Joseph, N., DasSarma, N., Henighan, T., Mann, B., Askell, A., Bai, Y., Chen, A., Conerly, T., Drain, D., Ganguli, D., Hatfield-Dodds, Z., Hernandez, D., Johnston, S., Jones, A., Kernion, J., Lovitt, L., Ndousse, K., Amodei, D., Brown, T., Clark, J., Kaplan, J., McCandlish, S., and Olah, C. (2022). In-context learning and induction heads. *Transformer Circuits Thread*. https://transformer-circuits.pub/2022/in-context-learning-and-induction-heads/index.html.

Ortu, F., Jin, Z., Doimo, D., Sachan, M., Cazzaniga, A., and Schölkopf, B. (2024). Competition of mechanisms: Tracing how language models handle facts and counterfactuals.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library.

Pearl, J. (2013). Direct and indirect effects.

Periti, F., Dubossarsky, H., and Tahmasebi, N. (2024). (chat)GPT v BERT dawn of justice for semantic change detection. In Graham, Y. and Purver, M., editors, *Findings of the Association for Computational Linguistics: EACL 2024*, pages 420–436, St. Julian's, Malta. Association for Computational Linguistics.

Quionero-Candela, J., Sugiyama, M., Schwaighofer, A., and Lawrence, N. (2009). Dataset shift in machine learning.

Radford, A. and Narasimhan, K. (2018). Improving language understanding by generative pre-training.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. (2019). Language models are unsupervised multitask learners.

Ren, J., Yu, C., Sheng, S., Ma, X., Zhao, H., Yi, S., and Li, H. (2020). Balanced meta-softmax for long-tailed visual recognition.

Rogers, A., Kovaleva, O., and Rumshisky, A. (2020). A primer in bertology: What we know about how bert works.

Saenko, K., Kulis, B., Fritz, M., and Darrell, T. (2010). Adapting visual category models to new domains. volume 6314, pages 213–226.

Sagawa, S., Koh, P. W., Hashimoto, T. B., and Liang, P. (2020). Distributionally robust neural networks for group shifts: On the importance of regularization for worst-case generalization.

Sanh, V., Debut, L., Chaumond, J., and Wolf, T. (2020). Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter.

Sinitsin, A., Plokhotnyuk, V., Pyrkin, D., Popov, S., and Babenko, A. (2020). Editable neural networks.

Tatman, R. (2017). Gender and dialect bias in YouTube's automatic captions. In Hovy, D., Spruit, S., Mitchell, M., Bender, E. M., Strube, M., and Wallach, H., editors, *Proceedings of the First ACL Workshop on Ethics in Natural Language Processing*, pages 53–59, Valencia, Spain. Association for Computational Linguistics.

Thorne, J., Vlachos, A., Christodoulopoulos, C., and Mittal, A. (2018). Fever: a large-scale dataset for fact extraction and verification.

Vapnik, V. N. (1999). An overview of statistical learning theory. *IEEE transactions on neural networks*, 10 5:988–99.

Varma, V., Shah, R., Kenton, Z., Kramár, J., and Kumar, R. (2023). Explaining grokking through circuit efficiency.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2023). Attention is all you need.

Vig, J., Gehrmann, S., Belinkov, Y., Qian, S., Nevo, D., Singer, Y., and Shieber, S. (2020). Investigating gender bias in language models using causal mediation analysis. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 12388–12401. Curran Associates, Inc.

Voita, E. and Titov, I. (2020). Information-theoretic probing with minimum description length. In Webber, B., Cohn, T., He, Y., and Liu, Y., editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 183–196, Online. Association for Computational Linguistics.

Wang, K., Variengien, A., Conmy, A., Shlegeris, B., and Steinhardt, J. (2022). Interpretability in the wild: a circuit for indirect object identification in gpt-2 small.

Wen, K., Li, Y., Liu, B., and Risteski, A. (2023). Transformers are uninterpretable with myopic methods: a case study with bounded dyck grammars.

Williams, A., Nangia, N., and Bowman, S. (2018). A broad-coverage challenge corpus for sentence understanding through inference. In Walker, M., Ji, H., and Stent, A., editors, *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122, New Orleans, Louisiana. Association for Computational Linguistics.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Scao, T. L., Gugger, S., Drame, M., Lhoest, Q., and Rush, A. M. (2020). Huggingface's transformers: State-of-the-art natural language processing.

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., Klingner, J., Shah, A., Johnson, M., Liu, X., Łukasz Kaiser, Gouws, S., Kato, Y., Kudo, T., Kazawa, H., Stevens, K., Kurian, G., Patil, N., Wang, W., Young, C., Smith, J., Riesa, J., Rudnick, A., Vinyals, O., Corrado, G., Hughes, M., and Dean, J. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation.

Wu, Z., Geiger, A., Icard, T., Potts, C., and Goodman, N. D. (2024). Interpretability at scale: Identifying causal mechanisms in alpaca.

Yang, Y., Zhang, H., Katabi, D., and Ghassemi, M. (2023). Change is hard: A closer look at subpopulation shift.

Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R., and Le, Q. V. (2020). Xlnet: Generalized autoregressive pretraining for language understanding.

Yao, H., Wang, Y., Li, S., Zhang, L., Liang, W., Zou, J., and Finn, C. (2022). Improving out-of-distribution robustness via selective augmentation.

Yu, F., Zhang, H., Tiwari, P., and Wang, B. (2023). Natural language reasoning, a survey.

Zhang, F. and Nanda, N. (2024). Towards best practices of activation patching in language models: Metrics and methods.

Zhang, H., Cisse, M., Dauphin, Y. N., and Lopez-Paz, D. (2018). mixup: Beyond empirical risk minimization.

Zhang, K. and Bowman, S. (2018). Language modeling teaches you more than translation does: Lessons learned through auxiliary syntactic task analysis. In Linzen, T., Chrupała, G., and Alishahi, A., editors, *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 359–361, Brussels, Belgium. Association for Computational Linguistics.

Zhang, L., Wang, M., Chen, L., and Zhang, W. (2022). Probing GPT-3's linguistic knowledge on semantic tasks. In Bastings, J., Belinkov, Y., Elazar, Y., Hupkes, D., Saphra, N., and Wiegreffe, S., editors, *Proceedings of the Fifth BlackboxNLP Workshop on Analyzing and Interpreting Neural Networks for NLP*, pages 297–304, Abu Dhabi, United Arab Emirates (Hybrid). Association for Computational Linguistics.

Zhong, Z., Liu, Z., Tegmark, M., and Andreas, J. (2023). The clock and the pizza: Two stories in mechanistic explanation of neural networks.

Zhou, Z., Tarzanagh, D. A., Hou, B., Tong, B., Xu, J., Feng, Y., Long, Q., and Shen, L. (2023). Fair canonical correlation analysis.

Zhu, C., Rawat, A. S., Zaheer, M., Bhojanapalli, S., Li, D., Yu, F., and Kumar, S. (2020). Modifying memories in transformer models.

Appendix A

Supplemental Material

A.1 Suggestive Evidence for CRH

	Contradiction	Entailment	Neutral	Average
Contradiction Logit	1.12	0.65	1.22	1.00
Entailment Logit	-0.76	-0.59	-0.54	-0.63
Neutral Logit	-0.44	0.02	-0.77	-0.40

Table A.1 Average Logit Difference per Label. The difference is calculated by subtracting the first three columns in table 3.2 from the last three columns. The 4th column in the table is the average of the first three columns. Notably, for examples that have a "neutral" label, the presence of a shortcut attribute increases the contradiction logit 1.22 points on average. This increase in logits is higher than the logit increase in the contradiction examples, which is 1.12.

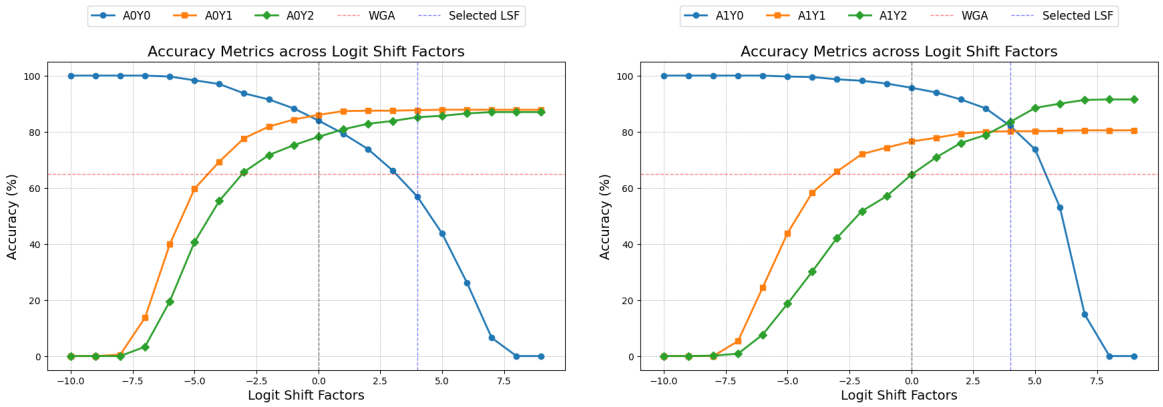
A.2 REPRSHIFT Implementation Details

A.2.1 Internal Representation for the Shortcut Attribute

To compute an internal representation for the shortcut attribute, we sample examples that contain shortcut attributes from the validation split of MultiNLI. In our experiments, we randomly sample a total of 1800 examples from the validation set, 600 examples from data groups a_1y_0, a_1y_1, and a_1y_2. Due to the low count of examples for the minority groups, 600 is around the maximum number of examples we can sample for certain groups.

A.2.2 Logit Shift

Our goal is to select a logit shift vector, $v_{logitshift}$, such that if $v_{logitshift}$ is added to the logits, it suppresses the effect of the shortcut rule on the logits. We select the logit shift factor via grid search. We search for a $d_{\text{labels}} = 3$ dimensional vector, $v_{logitshift}$, such that if $v_{logitshift}$ is added to the logits of the examples with shortcut attributes, the resulting predicted labels has best worst-group accuracy on the validation set. Figure A.1 demonstrates the worst group



(a) Examples with No Shortcut Attribute

(b) Examples with Shortcut Attribute

Fig. A.1 Data Group Accuracy across different logit shift factors. The x-axis represents the negative shift of the contradiction logits. The figure on the left shows the group accuracies for examples without shortcuts. We observe that, in the absence of the shortcut token, as we start shifting down the logits of the contradiction class (shift factor > 0), the accuracy of the examples with contradiction labels decreases sharply while the accuracies of the other groups increase slowly. We note that it is possible to achieve worst-group accuracy close to 80% worst-group accuracy on the examples that contain shortcut attributes by simply shifting the contradiction logits down 4 logit units. However, the same logit shift leads to performance drops on examples that do not have shortcut attributes.

accuracies across different one-dimensional logit shift factors for the contradiction label. In our experiments, we find that the logit shift factors that achieve best worst-group accuracies either shift down the contradiction logits or shift up the logits for the other two labels an equal amount. In our implementation, we shift down the contradiction logits and fix the other two components. This is consistent with the patterns observed in Table A.1.

A.2.3 Representation Shift Vector

In our implementation, we use the 3600 examples we sampled from the validation set, which contains 600 examples from each data group. 600 is near the maximum number of examples we can sample for the minority group. We note that in larger datasets, bigger samples would give more reliable estimates. We use full-batch gradient descent with Adam optimizer. We set the learning rate to 0.001 and do 1000 parameter updates to select optimum $v_{reprshift}$.

A.2.4 Modifying Model Weights

To compute the columns k_i of the keys matrix K, we save the internal representations, s_{CLS} (defined in Section 3.3.2) from a group balanced sample generated by 3600 examples from the validation set. This sample contains 600 examples from each of the 6 data groups.

A.3 Fine Tuning Setup

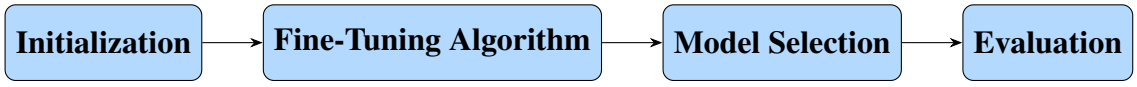


Fig. A.2 Summary of Fine-Tuning Setup

A.3.1 Initialization

We initialize our model with the pre-trained weights of the BERT-base model from Devlin et al. (2019). The classifier is the only part of our model that is randomly initialized.

A.3.2 Fine-Tuning Algorithm

We fine-tune each of the 3 initialized models with 5 different learning algorithms: Empirical Risk Minimization (Section 2.2.1), GroupDRO (Section 2.2.3), Focal Loss (Section 2.2.4), JTT (Section 2.2.5), and LFF (Section 2.2.6). We train our model for $30,000$ parameter update steps with a batch size of 32, a learning rate of 10^{-5}, and classifier dropout set to 0.5 following Yang et al. (2023). To train our models, we use we use the AdamW optimizer following the previous work (Idrissi et al., 2022; Izmailov et al., 2022; Yang et al., 2023). We randomize the order in which training data is seen during training and save 30 model checkpoints every $1,000$ update steps. In total, we fine-tune 30 models, 15 for each of our

datasets. For a given dataset, for each fine-tuning algorithm, we do 3 training runs, which we then use to compute the means and standard deviations in our experiments.

A.3.3 Model Selection

We use the validation set to compute the overall accuracy and group accuracies for the 30 training checkpoints of our 30 models. Unless stated otherwise, we select the model with the best worst group accuracy on the validation set. In Section 4.4.1 and when applying REPRSHIFT, we change our model selection procedure for the ERM-trained models and use the model with the best validation accuracy as our final model. We highlight that this does not lead to better worst-group accuracies on the test set. The motivation behind this design decision is two-fold: (1) In causal intervention experiments, we want to understand the shortcut learning behavior in the models that have the best generalization properties on IID test sets. (2) Using REPRSHIFT, we want to remove the shortcut learning behavior in the models that have the best generalization properties on IID test sets.

A.3.4 Hyperparameter Selection

Name	Hyperparameters	Search Space
Group DRO	$\eta_{DRO} = 0.01$	$10^{\text{Uniform(-3,-1)}}$
Focal Loss	$\gamma = 1$	$0.5 * 10^{\text{Uniform(0,1)}}$
JTT	First Step Fraction $= \frac{1}{2}$ $\lambda_{JTT} = 10$	$\text{Uniform}(0.2, 0.8)$ $10^{\text{Uniform(0,2.5)}}$
LFF	$\mathfrak{q} = 0.7$	$\text{Uniform}(0.05, 0.95)$

Table A.2 Selected Hyperparameters

Table A.2 shows the hyperparameters we used for our learning algorithms and the corresponding hyperparameter search spaces. Yang et al. (2023) ran 16 random trials over a joint distribution of the hyperparameters for each algorithm and identified the best hyperparameters using the validation set. We do not repeat this procedure as it requires fine-tuning 128 models for our two datasets. Instead, we adopt the algorithm-specific hyper-parameters selected by Yang et al. (2023). We fix these hyperparameters across three training runs in all of our experiments. Gulrajani and Lopez-Paz (2020) have argued that this procedure ensures a fair "best-versus-best" comparison between different algorithms.

A.4 Examples from Datasets

Attribute	Label	Premise	Hypothesis
	Contradiction $y = 0$	privatization assumes an end to the ponzi every generation saves for its own retirement.	privatization has ended the ponzi and generations no longer save for retirement.
Negation Word $a = 1$	Entailment $y = 1$	only the right arm of one of the two transepts and the octagonal belltower [...] remain.	the left arms of the two transepts no longer remain.
	Neutral $y = 2$	uh yeah except i don't use them	i have never found the need to use them.
	Contradiction $y = 0$	the museum of traditional industry presents a diverse collection of textiles, porcelain, fans, dolls,[...]	the museum of traditional industry is completely dedicated to textiles.
No Negation Word $a = 0$	Entailment $y = 1$	some of the programs'managers stayed with one program for their entire professional lifetimes.	some of the program managers only worked on one program their entire career.
	Neutral $y = 2$	[...] but they think and probably the right people buy it.	they think this.

Table A.3 Examples from MultiNLI

Attribute	Label	Comment
Male $a = 0$	Non-toxic $y = 0$	"had the opportunity to watch this remarkable man [...]"
	Toxic $y = 1$	"men like miller seek proximity to supposed alpha males like trump and bannon [...]"
Female $a = 1$	Non-toxic $y = 0$	"if you have to issue a statement that you're not'anti-women [...] your anti-women."
	Toxic $y = 1$	"punch a lady in the face! [...]"
LGBTQ $a = 2$	Non-toxic $y = 0$	"[...] adultery is probably more serious (my opinion) than homosexuality [...]"
	Toxic $y = 1$	"why equal immigration just with welfare, crime and terrorism? [...] overall [they're] determined to rape our daughters [...](they like homosex, you know). [...]"
Christian $a = 3$	Non-toxic $y = 0$	"[...] anyone who resists change inconsistent with scriptural revelation and established catholic morality?"
	Toxic $y = 1$	"[...] the current catholic sexual ethic is asexuals doing self-preservation."
Muslim $a = 4$	Non-toxic $y = 0$	"the protest did prompt nishika ramaswami to hold a sign supporting muslims."
	Toxic $y = 1$	"after muslim killer kills 6 in quebec sobbering leftist zombies wobble towards reality [...]"
Other Religions $a = 5$	Non-toxic $y = 0$	"[...] people claiming to be "devout" agnostics or atheists. [...]"
	Toxic $y = 1$	"[...] atheists believe that what looks like a painting has always existed and no one painted it. [...] they will be cast into hell [...]"
Black $a = 6$	Non-toxic $y = 0$	"the battle for equality is ongoing. [...] lincoln did not at all always embrace equality of the black person." [...]
	Toxic $y = 1$	"we wouldn't want to say they were black, wouldn't be politically correct."
White $a = 7$	Non-toxic $y = 0$	"so a white supremacist group that preaches anti-semitism [...] would not be a hate group [...] ?"
	Toxic $y = 1$	"[...] lets just get rid of these bums who are not white."

Table A.4 Examples from CivilComments

A.5 Probing Setup

We randomly split the representations 3600 representations from a fine-tuning method for a particular layer into training (70%), validation (15%), and test (15%) splits. Using the training set, we train one classifier that predicts the shortcut attributes from the representations and another classifier that predicts the true labels from the representations. Prior to training, we normalize the representations so that the features have mean 0 and standard deviation 1. We train the probes for 100 epochs with full-batch gradient descent and cross-entropy loss. We use Adam optimizer with a learning rate of 0.001. Finally, the validation split is used to select the best model. We evaluate the probe accuracy on the test split. Our MLP probe has a hidden dimension of 128.

A.6 Resources

We trained our models using PyTorch (Paszke et al., 2019). For different loss function-based approaches, we based our code on the implementations from Yang et al. (2023). We accessed the pre-trained weights of the BERT-base model via Huggingface (Wolf et al., 2020). After training our models, we loaded our trained-model weights to the HookedEncoder model from TransformerLens (Nanda and Bloom, 2022) for our experiments. HookedEncoder did not have support for BERT for classification objective but supported BERT for masked language modeling (MLM) objective. With minor adjustments, we adapted the BERT for MLM implementation for our classification tasks. We implemented the rank-one update based on Meng et al. (2023a, Equation 2). For our experiments, we used NVIDIA A100 GPUs.

Appendix B

Additional Results

	ERM	GroupDRO	Focal	JTT	LFF
Overall	81.2 (0.6)	81.1 (0.2)	81.1 (0.3)	80.7 (0.1)	59.0 (2.7)
WGA	64.6 (1.1)	64.3 (1.8)	65.6 (2.2)	64.3 (2.3)	46.1 (13.4)
EWA	79.8 (0.1)	79.3 (0.2)	79.6 (0.4)	78.9 (0.0)	58.1 (2.9)
y=0, a=0	79.2 (1.2)	80.1 (0.5)	79.2 (0.6)	78.1 (0.4)	58.3 (16.5)
y=0, a=1	94.5 (0.8)	94.2 (0.2)	94.7 (0.5)	94.8 (0.4)	76.6 (13.1)
y=1, a=0	83.2 (0.9)	81.1 (0.9)	82.0 (1.4)	81.9 (0.8)	52.1 (16.6)
y=1, a=1	77.0 (1.3)	73.6 (1.7)	75.0 (0.3)	74.8 (1.0)	46.1 (13.4)
y=2, a=0	79.3 (0.5)	80.3 (1.4)	80.2 (0.5)	80.1 (0.4)	64.0 (6.2)
y=2, a=1	64.6 (1.1)	64.3 (1.8)	65.6 (2.2)	64.3 (2.3)	62.0 (10.7)

Table B.1 MultiNLI Test Set Detailed Results.

	ERM	GroupDRO	Focal	JTT	LFF
Overall	84.8 (1.2)	80.7 (0.9)	84.9 (0.5)	83.5 (1.7)	45.4 (21.4)
WGA	66.2 (3.2)	72.8 (3.0)	63.4 (0.2)	69.6 (4.1)	14.5 (2.0)
EWA	77.5 (0.3)	78.8 (0.5)	76.5 (0.1)	78.1 (0.4)	50.3 (19.1)
y=0,a=0	89.7 (1.8)	81.8 (2.1)	90.8 (0.9)	87.9 (3.2)	41.6 (25.1)
y=0,a=1	90.8 (1.8)	82.2 (2.0)	91.9 (0.6)	89.4 (2.7)	44.4 (27.9)
y=0,a=2	80.9 (4.0)	73.5 (2.5)	81.1 (1.2)	76.7 (4.7)	19.6 (6.4)
y=0,a=3	93.2 (1.2)	87.5 (0.5)	93.8 (0.4)	92.2 (1.6)	51.1 (33.8)
y=0,a=4	83.1 (4.0)	78.3 (1.0)	84.8 (1.2)	81.0 (3.2)	29.9 (15.8)
y=0,a=5	89.1 (2.0)	83.4 (0.6)	88.5 (0.8)	86.8 (2.5)	38.0 (22.8)
y=0,a=6	81.0 (2.8)	72.8 (3.0)	78.6 (2.7)	74.4 (5.6)	14.5 (2.0)
y=0,a=7	81.1 (3.1)	74.9 (3.3)	79.4 (1.8)	74.5 (5.5)	15.0 (2.3)
y=1,a=0	70.4 (2.5)	81.1 (3.3)	67.9 (1.4)	73.4 (6.0)	78.9 (19.2)
y=1,a=1	69.7 (2.9)	82.4 (3.4)	67.0 (1.3)	72.5 (5.4)	78.7 (17.2)
y=1,a=2	68.3 (4.2)	76.7 (4.0)	67.3 (0.8)	73.1 (5.0)	83.1 (19.2)
y=1,a=3	66.2 (3.2)	80.4 (1.3)	63.4 (0.2)	69.6 (4.1)	78.3 (13.7)
y=1,a=4	67.2 (4.5)	74.2 (1.5)	65.7 (2.2)	71.9 (2.8)	81.1 (18.5)
y=1,a=5	68.9 (3.1)	76.1 (0.9)	66.0 (1.8)	70.4 (3.4)	80.9 (17.5)
y=1,a=6	69.4 (1.7)	78.5 (2.9)	71.2 (1.8)	76.4 (5.1)	83.0 (22.0)
y=1,a=7	69.8 (2.7)	76.8 (3.8)	70.0 (1.7)	75.5 (5.3)	81.9 (23.6)

Table B.2 CivilComments Test Set Detailed Results.

	ERM	GroupDRO	Focal	JTT	LFF
Overall	81.6 (0.4)	81.1 (0.2)	81.2 (0.2)	80.8 (0.2)	51.3 (8.3)
WGA	66.5 (1.3)	64.9 (1.3)	66.9 (1.6)	64.7 (2.4)	45.8 (21.5)
EWA	71.6 (0.3)	70.5 (0.4)	70.4 (0.0)	69.2 (0.6)	47.0 (5.5)
y=0, a=0	80.1 (0.8)	79.9 (0.3)	78.8 (0.1)	78.7 (0.0)	53.5 (18.3)
y=0, a=1	95.0 (0.8)	94.3 (0.2)	94.4 (0.4)	94.8 (0.6)	48.6 (12.2)
y=1, a=0	83.9 (0.5)	81.6 (0.9)	82.9 (1.4)	81.9 (0.9)	45.8 (21.5)
y=1, a=1	75.3 (1.9)	73.7 (0.7)	73.0 (0.4)	72.8 (0.6)	62.0 (4.6)
y=2, a=0	78.8 (0.9)	79.9 (1.8)	79.8 (0.5)	79.7 (0.4)	54.9 (17.4)
y=2, a=1	66.5 (1.3)	64.9 (1.3)	66.9 (1.6)	64.7 (2.4)	62.4 (10.2)

Table B.3 MultiNLI Validation Set Detailed Results.

	ERM	GroupDRO	Focal	JTT	LFF
Overall	84.2 (0.6)	81.2 (1.0)	84.5 (0.2)	83.2 (1.4)	39.7 (22.1)
WGA	63.3 (2.1)	71.9 (1.5)	57.4 (1.4)	66.7 (6.2)	17.6 (3.5)
EWA	72.5 (0.2)	74.0 (0.0)	71.7 (0.2)	74.2 (0.9)	46.1 (18.9)
y=0, a=0	89.6 (1.5)	83.2 (2.3)	91.0 (0.7)	87.6 (2.6)	37.0 (24.5)
y=0, a=1	89.8 (1.1)	82.8 (1.8)	91.3 (0.3)	88.0 (2.4)	37.8 (26.7)
y=0, a=2	79.9 (1.0)	73.9 (2.7)	80.6 (0.9)	77.1 (5.3)	24.3 (2.3)
y=0, a=3	93.8 (0.5)	88.8 (0.6)	94.4 (0.1)	92.7 (1.7)	42.4 (36.3)
y=0, a=4	80.5 (3.1)	79.5 (0.8)	83.6 (1.3)	80.1 (3.2)	29.2 (14.8)
y=0, a=5	87.3 (1.3)	84.0 (0.2)	89.1 (0.0)	86.2 (3.7)	35.5 (24.2)
y=0, a=6	77.5 (0.0)	72.2 (3.7)	76.0 (1.3)	71.1 (4.3)	17.8 (4.5)
y=0, a=7	78.8 (1.4)	75.0 (3.6)	78.7 (2.4)	72.2 (4.5)	17.6 (3.5)
y=1, a=0	74.5 (3.4)	83.1 (3.3)	70.4 (1.0)	78.5 (5.7)	64.5 (29.6)
y=1, a=1	70.2 (2.9)	81.6 (3.0)	65.1 (1.3)	74.4 (5.2)	66.1 (24.1)
y=1, a=2	67.5 (1.8)	73.6 (3.5)	63.7 (2.0)	72.9 (7.0)	68.6 (27.2)
y=1, a=3	63.3 (2.1)	75.9 (1.2)	57.4 (1.4)	67.2 (6.5)	67.6 (17.8)
y=1, a=4	68.8 (3.0)	72.3 (1.2)	62.6 (3.8)	71.2 (4.0)	69.3 (22.9)
y=1, a=5	66.4 (4.0)	71.9 (1.5)	58.6 (3.1)	66.7 (6.2)	65.7 (23.2)
y=1, a=6	68.6 (0.3)	77.8 (3.3)	73.0 (3.2)	78.2 (4.7)	66.1 (32.6)
y=1, a=7	70.1 (2.7)	73.9 (3.9)	67.8 (3.2)	76.6 (5.5)	66.5 (31.4)

Table B.4 CivilComments Validation Set Detailed Results.

	Random	Random Init	Pretrained	ERM	GroupDRO	Focal	JTT	LFF
Shortcut Attribute Probe - Linear								
Layer 1	49.9 (2.1)	82.0 (0.8)	84.5 (0.5)	85.2 (0.4)	83.5 (1.4)	85.4 (0.7)	84.6 (1.0)	85.3 (0.6)
Layer 2	50.3 (0.8)	81.7 (1.7)	84.7 (0.3)	85.7 (0.9)	84.3 (0.6)	85.7 (0.4)	85.4 (1.1)	84.3 (0.4)
Layer 3	48.7 (0.4)	78.9 (1.6)	85.4 (1.5)	86.8 (1.5)	86.5 (1.0)	86.0 (1.0)	84.7 (0.7)	85.5 (1.1)
Layer 4	52.4 (2.1)	77.0 (1.3)	79.6 (2.5)	86.3 (0.7)	86.5 (0.5)	87.2 (0.3)	86.7 (0.8)	85.3 (0.5)
Layer 5	51.6 (0.8)	75.7 (2.1)	80.0 (2.5)	87.1 (1.0)	87.3 (1.9)	87.5 (1.1)	86.1 (2.0)	85.1 (1.2)
Layer 6	49.2 (2.0)	73.4 (3.0)	84.5 (1.2)	88.8 (2.8)	90.0 (1.7)	89.2 (1.3)	90.2 (1.8)	90.0 (1.1)
Layer 7	51.1 (1.5)	72.0 (3.4)	83.1 (0.8)	89.0 (2.0)	90.6 (1.6)	89.3 (1.3)	90.2 (1.6)	90.6 (1.2)
Layer 8	51.4 (0.5)	80.9 (7.1)	83.5 (0.2)	89.0 (1.4)	91.3 (1.2)	90.3 (1.7)	91.0 (1.4)	92.2 (1.7)
Layer 9	52.7 (2.2)	96.2 (0.8)	84.9 (2.0)	89.2 (0.2)	92.2 (0.2)	89.2 (0.6)	90.9 (0.4)	93.0 (2.3)
Layer 10	48.5 (1.4)	97.9 (0.6)	83.1 (0.9)	90.5 (0.5)	91.0 (0.8)	90.9 (0.2)	90.0 (0.7)	92.4 (1.5)
Layer 11	50.0 (3.7)	98.1 (0.2)	82.2 (1.6)	89.1 (0.2)	90.1 (0.4)	89.0 (0.7)	89.4 (0.7)	91.8 (1.3)
Layer 12	49.3 (1.0)	98.0 (0.3)	78.2 (0.8)	88.3 (0.4)	88.9 (0.8)	88.3 (0.6)	89.1 (0.2)	90.9 (1.5)
Shortcut Attribute Probe - 2 Layer MLP								
Layer 1	51.2 (2.9)	82.2 (1.5)	84.6 (0.5)	84.8 (1.6)	83.5 (0.3)	84.0 (0.9)	84.3 (1.4)	84.3 (1.3)
Layer 2	49.5 (1.5)	80.4 (0.8)	86.5 (1.4)	87.5 (0.8)	85.7 (0.8)	86.4 (0.6)	87.2 (1.5)	86.7 (1.6)
Layer 3	49.4 (1.5)	79.9 (1.0)	86.1 (1.1)	87.2 (1.1)	87.0 (2.0)	86.4 (1.6)	85.9 (0.8)	86.0 (0.9)
Layer 4	52.3 (1.8)	77.8 (1.3)	81.6 (3.0)	87.0 (2.0)	86.9 (1.2)	86.9 (0.7)	87.2 (0.8)	83.8 (1.1)
Layer 5	51.2 (0.9)	75.0 (2.2)	81.0 (2.7)	88.6 (1.2)	89.0 (1.5)	88.2 (1.8)	87.4 (2.6)	86.3 (1.5)
Layer 6	49.3 (1.6)	73.3 (2.0)	84.1 (1.5)	90.0 (1.9)	91.5 (1.2)	90.3 (0.9)	91.5 (1.7)	90.5 (0.9)
Layer 7	51.4 (2.5)	74.3 (3.7)	83.3 (0.7)	90.7 (1.6)	91.7 (2.0)	89.8 (0.8)	90.6 (1.7)	91.4 (1.7)
Layer 8	51.8 (1.1)	86.0 (6.3)	84.9 (1.4)	90.3 (1.8)	92.3 (1.5)	90.8 (1.5)	90.7 (1.8)	92.6 (2.0)
Layer 9	53.3 (2.1)	97.0 (0.8)	85.6 (1.5)	91.5 (0.2)	93.1 (0.7)	91.2 (0.6)	91.5 (1.3)	93.7 (2.4)
Layer 10	49.1 (1.2)	98.1 (0.7)	84.5 (1.2)	92.1 (0.4)	93.1 (0.4)	91.7 (0.3)	92.0 (0.8)	92.9 (2.1)
Layer 11	50.5 (3.7)	98.5 (0.1)	82.5 (1.5)	90.2 (0.5)	92.4 (0.2)	90.6 (0.5)	90.7 (1.3)	92.8 (2.2)
Layer 12	49.9 (1.1)	98.4 (0.5)	77.7 (0.9)	90.1 (0.4)	90.9 (0.8)	89.6 (0.7)	89.3 (1.3)	91.9 (1.7)
Y Label Probe - Linear								
Layer 1	33.5 (2.0)	42.3 (2.9)	41.4 (1.1)	42.7 (2.4)	42.2 (2.5)	42.1 (2.7)	41.5 (1.2)	41.7 (1.7)
Layer 2	32.8 (1.7)	40.4 (3.0)	41.4 (2.5)	42.0 (2.2)	40.5 (1.8)	40.6 (1.8)	40.6 (2.8)	41.0 (2.2)
Layer 3	31.8 (0.6)	41.4 (3.2)	46.8 (0.9)	47.4 (2.4)	48.5 (1.4)	47.5 (3.0)	47.3 (1.7)	47.3 (1.7)
Layer 4	33.0 (1.5)	39.6 (1.2)	47.9 (2.4)	52.2 (1.1)	52.5 (1.4)	52.5 (1.1)	52.5 (1.7)	50.1 (0.6)
Layer 5	34.7 (1.7)	39.1 (2.7)	50.7 (1.3)	56.9 (2.3)	58.1 (1.3)	57.3 (0.5)	56.9 (0.7)	55.7 (0.6)
Layer 6	33.3 (1.2)	39.3 (1.9)	53.6 (0.9)	65.9 (0.5)	66.0 (0.9)	64.6 (0.3)	66.5 (0.5)	64.9 (1.4)
Layer 7	34.2 (0.8)	39.1 (2.8)	56.1 (1.2)	69.5 (0.9)	69.3 (0.8)	69.6 (0.2)	69.3 (1.3)	66.9 (0.8)
Layer 8	32.4 (1.3)	41.7 (1.2)	56.1 (1.2)	71.7 (0.3)	70.9 (0.3)	71.8 (0.9)	71.9 (1.6)	67.7 (1.6)
Layer 9	32.0 (2.5)	48.6 (2.2)	57.5 (2.5)	76.5 (0.5)	76.5 (0.6)	76.0 (1.6)	77.1 (0.9)	69.5 (0.1)
Layer 10	32.3 (2.5)	44.9 (1.6)	56.4 (2.9)	78.4 (1.2)	77.8 (0.6)	77.7 (0.7)	78.1 (0.8)	70.2 (0.7)
Layer 11	33.9 (1.3)	45.3 (1.5)	55.1 (2.3)	79.0 (0.2)	78.1 (0.3)	78.5 (0.6)	78.1 (0.4)	70.9 (0.1)
Layer 12	32.7 (2.4)	45.2 (1.1)	52.4 (0.5)	80.2 (0.2)	78.8 (0.9)	78.4 (1.8)	79.7 (0.5)	70.4 (0.5)
Y Label Probe - 2 Layer MLP								
Layer 1	34.0 (2.6)	39.0 (1.4)	41.0 (0.7)	41.2 (1.5)	39.8 (1.7)	40.8 (1.0)	41.2 (0.7)	40.7 (1.8)
Layer 2	33.4 (2.0)	38.3 (1.3)	39.9 (3.0)	40.6 (3.0)	42.3 (3.7)	41.5 (4.4)	40.1 (3.3)	40.1 (3.4)
Layer 3	32.2 (1.3)	39.4 (2.5)	42.3 (1.1)	47.2 (2.1)	46.9 (2.5)	45.5 (3.6)	45.3 (2.7)	44.6 (1.5)
Layer 4	32.2 (1.7)	39.2 (3.5)	45.1 (4.2)	49.4 (0.9)	51.5 (1.7)	51.4 (2.1)	50.2 (4.0)	48.5 (1.4)
Layer 5	33.6 (1.1)	37.8 (2.4)	48.6 (1.1)	57.8 (1.7)	57.7 (1.1)	57.7 (0.6)	56.5 (1.3)	54.8 (1.3)
Layer 6	34.0 (0.4)	36.5 (2.9)	55.1 (1.8)	66.5 (0.5)	66.7 (1.4)	65.6 (0.9)	66.4 (1.1)	64.4 (1.6)
Layer 7	33.9 (1.2)	38.0 (1.9)	55.1 (1.2)	69.8 (1.1)	68.8 (1.1)	70.2 (1.3)	69.6 (0.6)	67.2 (1.9)
Layer 8	32.8 (1.1)	40.8 (1.5)	56.4 (0.2)	71.4 (0.6)	71.9 (0.5)	70.3 (0.8)	71.1 (0.9)	67.0 (2.1)
Layer 9	32.5 (1.3)	44.6 (1.8)	57.0 (1.3)	76.3 (0.7)	75.0 (0.4)	75.2 (1.3)	75.6 (0.2)	69.3 (1.5)
Layer 10	31.6 (0.8)	44.7 (1.4)	54.3 (2.4)	77.0 (0.9)	76.7 (0.4)	76.9 (1.6)	77.2 (0.6)	69.1 (0.2)
Layer 11	34.5 (2.3)	47.1 (1.6)	53.4 (2.4)	77.3 (0.5)	77.6 (0.7)	77.5 (0.7)	77.3 (1.1)	70.4 (1.3)
Layer 12	33.4 (2.0)	44.8 (1.5)	50.9 (3.1)	79.2 (1.4)	78.7 (1.3)	78.4 (0.9)	77.8 (0.8)	69.6 (0.9)

Table B.5 Probes on MultiNLI.

	Random	Random Init	Pretrained	ERM	GroupDRO	Focal	JTT	LFF
Shortcut Attribute Probe - Linear								
Layer 1	13.4 (1.1)	43.4 (2.7)	42.4 (1.4)	43.1 (1.6)	42.7 (1.0)	41.0 (1.7)	42.4 (1.2)	43.6 (2.2)
Layer 2	11.4 (0.9)	40.7 (0.7)	46.4 (1.0)	46.9 (3.0)	47.8 (1.5)	46.9 (0.7)	46.9 (2.1)	47.8 (2.9)
Layer 3	12.3 (0.7)	37.6 (0.3)	53.3 (1.6)	54.5 (1.0)	54.0 (1.0)	54.9 (1.4)	56.5 (1.9)	53.0 (1.2)
Layer 4	12.5 (1.7)	38.2 (0.3)	56.9 (1.8)	58.0 (2.1)	56.8 (1.1)	55.1 (2.8)	57.8 (1.6)	57.0 (1.8)
Layer 5	13.5 (1.2)	36.1 (0.2)	54.0 (1.2)	56.7 (2.9)	53.7 (1.6)	54.1 (2.4)	57.9 (1.1)	57.1 (1.7)
Layer 6	11.9 (0.6)	33.6 (0.6)	49.9 (1.3)	54.0 (1.1)	51.2 (1.7)	49.0 (3.4)	52.6 (0.7)	55.0 (0.9)
Layer 7	12.0 (0.9)	33.0 (1.7)	42.7 (1.7)	48.3 (2.5)	45.1 (1.8)	44.6 (3.7)	46.5 (1.7)	50.8 (0.7)
Layer 8	13.3 (0.4)	31.7 (2.0)	39.7 (1.4)	44.9 (3.6)	42.6 (3.3)	42.7 (0.6)	44.5 (1.3)	50.9 (2.4)
Layer 9	12.7 (1.4)	27.3 (2.6)	36.5 (2.2)	43.9 (3.3)	42.4 (1.6)	40.4 (4.0)	43.1 (4.2)	49.9 (4.0)
Layer 10	13.1 (0.3)	21.0 (2.1)	33.5 (1.1)	46.2 (2.0)	44.9 (2.4)	43.5 (1.1)	46.4 (1.7)	50.8 (7.6)
Layer 11	12.0 (0.5)	19.2 (1.3)	35.8 (2.0)	56.2 (2.1)	53.8 (3.0)	54.3 (0.8)	55.0 (2.1)	54.0 (8.5)
Layer 12	13.1 (0.5)	18.0 (0.3)	51.2 (1.8)	59.8 (2.3)	58.1 (1.5)	58.5 (1.8)	59.7 (1.2)	54.1 (10.2)
Shortcut Attribute Probe - 2 Layer MLP								
Layer 1	12.0 (0.9)	39.3 (2.8)	39.0 (1.5)	38.1 (1.1)	39.8 (1.1)	36.7 (0.5)	39.0 (2.8)	39.9 (1.9)
Layer 2	11.6 (0.9)	36.9 (0.7)	42.4 (2.5)	43.4 (1.7)	43.8 (0.7)	44.4 (1.4)	44.3 (2.9)	44.1 (2.3)
Layer 3	11.3 (1.2)	34.0 (0.6)	50.6 (1.6)	53.4 (2.9)	52.8 (2.0)	52.5 (2.7)	53.6 (0.8)	52.8 (1.2)
Layer 4	13.5 (2.9)	34.1 (0.9)	53.7 (1.4)	55.9 (1.9)	53.3 (1.8)	53.1 (3.2)	55.0 (0.8)	55.0 (1.1)
Layer 5	12.9 (1.8)	29.7 (0.2)	53.3 (1.6)	55.5 (2.2)	53.1 (1.2)	50.2 (2.6)	55.0 (1.4)	56.7 (1.2)
Layer 6	13.2 (1.5)	29.1 (1.4)	48.1 (2.2)	52.0 (2.3)	48.0 (1.9)	47.1 (3.9)	51.0 (1.3)	53.0 (2.4)
Layer 7	12.5 (1.0)	30.1 (1.2)	38.6 (3.1)	42.8 (3.0)	42.7 (1.9)	39.9 (5.5)	43.1 (1.5)	48.1 (1.8)
Layer 8	12.7 (0.2)	27.2 (1.3)	35.6 (2.1)	41.5 (5.7)	39.2 (2.6)	38.5 (2.4)	39.4 (2.6)	50.0 (2.0)
Layer 9	12.0 (1.4)	25.9 (1.1)	33.1 (1.5)	40.6 (3.9)	39.1 (2.5)	37.2 (4.0)	40.1 (2.4)	47.0 (4.0)
Layer 10	12.0 (1.2)	20.1 (0.4)	31.6 (0.3)	44.1 (2.3)	43.2 (3.0)	40.2 (2.7)	43.1 (2.9)	49.1 (5.7)
Layer 11	12.2 (1.1)	18.4 (2.4)	32.3 (1.2)	53.8 (2.2)	52.3 (3.2)	51.1 (1.3)	52.4 (0.8)	52.5 (8.0)
Layer 12	13.0 (0.3)	17.6 (1.0)	49.8 (1.8)	57.8 (3.4)	55.1 (1.4)	55.9 (1.7)	56.7 (2.2)	54.0 (8.3)
Y Label Probe - Linear								
Layer 1	50.0 (2.0)	60.9 (1.9)	65.8 (2.3)	67.5 (2.4)	67.9 (2.0)	67.8 (1.6)	66.5 (1.4)	67.2 (2.1)
Layer 2	48.3 (3.5)	61.4 (1.6)	69.2 (0.5)	71.3 (1.9)	70.2 (1.2)	70.3 (1.8)	70.4 (1.1)	69.4 (1.5)
Layer 3	49.9 (0.8)	62.0 (2.1)	69.9 (2.0)	72.7 (1.5)	73.0 (1.5)	72.3 (2.0)	72.5 (0.8)	71.4 (2.3)
Layer 4	48.6 (1.3)	60.4 (3.1)	70.1 (1.2)	71.9 (0.7)	72.7 (1.6)	70.7 (1.9)	72.8 (0.9)	72.6 (1.1)
Layer 5	50.4 (1.8)	61.0 (2.6)	70.3 (0.4)	73.7 (0.6)	73.5 (1.5)	71.3 (1.2)	72.9 (1.0)	71.0 (1.2)
Layer 6	49.0 (2.4)	60.7 (3.2)	68.8 (1.5)	72.5 (1.8)	74.6 (0.8)	71.6 (0.7)	73.6 (0.5)	73.1 (1.3)
Layer 7	50.6 (1.9)	60.8 (4.0)	67.8 (1.2)	72.5 (1.1)	73.6 (1.9)	71.1 (1.7)	72.8 (0.9)	71.6 (1.0)
Layer 8	49.4 (1.6)	70.9 (3.7)	68.0 (2.9)	74.0 (2.8)	77.7 (1.3)	74.1 (1.6)	74.1 (1.0)	73.5 (1.9)
Layer 9	51.7 (2.6)	76.7 (1.2)	68.4 (3.1)	74.9 (2.4)	77.5 (2.4)	74.1 (2.3)	77.0 (0.9)	75.1 (1.5)
Layer 10	51.2 (0.8)	76.0 (0.9)	67.2 (1.0)	77.1 (3.4)	79.5 (2.7)	76.7 (1.5)	78.3 (2.4)	75.6 (1.0)
Layer 11	51.5 (1.1)	73.2 (2.5)	68.6 (2.2)	78.3 (2.3)	79.7 (2.0)	79.3 (1.0)	78.1 (2.9)	75.9 (1.2)
Layer 12	48.1 (1.1)	74.7 (1.1)	68.5 (2.2)	78.8 (2.7)	80.2 (2.5)	78.3 (1.7)	79.0 (1.7)	76.3 (2.0)
Y Label Probe - 2 Layer MLP								
Layer 1	49.5 (2.3)	61.4 (0.6)	66.8 (3.1)	65.9 (2.6)	66.2 (2.1)	66.7 (2.1)	66.2 (2.3)	66.2 (1.9)
Layer 2	48.0 (3.8)	59.9 (1.6)	69.0 (1.7)	70.7 (2.5)	69.8 (3.0)	69.4 (2.8)	69.8 (2.0)	69.8 (1.7)
Layer 3	50.0 (0.7)	58.8 (2.0)	69.3 (2.5)	72.5 (2.3)	72.3 (2.6)	72.0 (2.1)	72.4 (2.1)	70.8 (2.3)
Layer 4	48.8 (0.9)	59.2 (1.4)	69.8 (1.9)	72.9 (0.6)	71.4 (2.7)	70.2 (3.4)	72.3 (1.5)	70.7 (1.7)
Layer 5	49.3 (1.3)	58.5 (1.0)	69.9 (0.6)	72.3 (0.6)	71.5 (1.1)	71.6 (0.8)	72.4 (0.8)	71.1 (1.9)
Layer 6	48.9 (1.2)	58.8 (1.7)	69.1 (0.3)	71.4 (0.3)	73.8 (0.4)	72.5 (0.2)	71.3 (1.2)	71.8 (2.0)
Layer 7	52.0 (2.6)	62.5 (2.5)	67.5 (1.8)	71.0 (2.1)	74.3 (1.6)	72.4 (2.3)	71.7 (1.1)	71.7 (0.7)
Layer 8	48.1 (0.4)	71.2 (2.2)	67.7 (1.2)	74.0 (2.4)	75.8 (1.8)	74.4 (1.8)	75.4 (1.2)	73.7 (1.4)
Layer 9	51.5 (3.5)	74.7 (2.8)	68.0 (1.8)	75.4 (1.8)	77.0 (1.4)	73.6 (2.7)	76.7 (1.3)	73.9 (0.9)
Layer 10	51.1 (1.6)	74.3 (1.8)	66.9 (2.7)	76.4 (1.7)	78.1 (1.3)	77.5 (2.3)	78.9 (2.0)	76.0 (1.1)
Layer 11	51.0 (1.0)	71.2 (1.2)	67.3 (0.9)	76.9 (2.5)	79.0 (1.2)	77.6 (0.8)	78.5 (1.4)	75.2 (1.4)
Layer 12	47.8 (0.5)	72.3 (2.2)	68.9 (1.9)	78.6 (2.4)	78.6 (2.6)	77.2 (1.5)	78.1 (1.6)	75.1 (1.6)

Table B.6 Probes on Civil Comments.